CHAPTER 2
WEB CACHING: TRENDS AND TECHNIQUES

2.1 Introduction

In web caching, frequently accessed cacheable objects can be stored in cache for accessing it in future instead of accessing it from web servers. A user requesting content served by the cache is able to avoid the delays inherent in the web such as slow servers and congested networks. Web caching systems lead to significant bandwidth savings, server load balancing, user perceived network latency reduction and higher content availability [39, 40]. Tauscher et al. reports that the probability of revisiting the web pages in web navigation is 58% [41]. In the web client proxy cache study, Duska et al. notes that up to 85% of cache hits were the result of multiple users requesting the same documents [42]. Some web resources such as constantly updated dynamic pages, resources that are personalized to a particular client etc. cannot be cached. Studies show that size of the static web pages increase approximately 15% per month [2]. This increases the possibility of caching. Fast access, robustness, transparency, scalability, efficiency, adaptivity, stability, load balancing, ability to deal with heterogeneity, simplicity are desirable properties of web caching system [43]. This chapter gives an overview of web caching and presents the survey in web caching.

The organization of this chapter is as follows. In section 2, web cache communication methods and cache sharing schemes are discussed. The benefits of web caching are discussed in section 3. Hyper Text Transfer Protocol (HTTP)
support for web caching is discussed in section 4. Different locations of cache servers are discussed in section 5. Various web caching protocols are introduced in section 6. Existing caching architectures/techniques are discussed in section 7. Different types of cache replacement strategies are discussed in section 8. Cache consistency mechanisms are outlined in section 9. The practical issues in the web cache design are discussed in section 10. Squeezing more performance from the partial information available in the web caches are discussed in section 11. Limitations of web caching are discussed in section 12.

2.2 Web Cache Communication

Web client is a browser, which has its own browser cache for providing immediate access to the objects stored in it. The proxy server accepts URL requests from clients, forwards that request to the web server, retrieves the requested information and sends it to the browser behind the firewall. In proxy caching, URL request from the client is forwarded through the proxy servers. The proxy servers accept the URL request from the client and it checks in its own local cache, if it is not available there, it forwards that request to the web server as in Figure 2.1.

![Figure 2.1 Caching Documents on a Proxy Server (Redrawn from [44])](image-url)
Upon receipt of that document, proxy server stores that document in its own cache. Then it sends that document to the client, which has requested. If an up-to-date version of the requested document is found in the cache of the proxy server, client request can be satisfied from the proxy’s local cache as in Figure 2.2. RAM and hard disk are used as cache for storing the cacheable objects. Accessing data from RAM is faster than hard disk, but cache size that could be assigned for web cache in main memory is less than hard disk. Markatos shown that even a small amount of main memory (512 Kbytes) that is used as a document cache is enough to hold more than 60% of the documents requested [45]. Combination of RAM and hard disk is preferable for using as web cache. Caches of proxy servers can be shared among cooperating proxy servers. The different cache sharing schemes [46] are given below.

- **No cache sharing**: Proxies do not collaborate to serve each other’s cache misses.

- **Simple cache sharing**: Once a proxy fetches a document from another proxy, it caches the document locally. Proxies do not coordinate cache replacements.

- **Single-copy cache sharing**: Proxies serve each other’s cache misses. Proxy does not cache documents fetched from another proxy but it marks the document
as most recently accessed. Storage of duplicate copies are eliminated and utilization of available cache space is increased.

- **Global cache**: Proxies fully cooperate for replacement and sharing. Proxy’s caches appear as one unified cache with global LRU replacement. In this scheme, it is assumed that all requests go to one cache and the size of the cache is the sum of all proxy cache sizes.

Caching policies can be applied at the level of individual user sessions, individual hosts and a collection of hosts on a LAN [47]. In session level, caches for separate sessions are managed independently. In host level, caches for separate hosts are managed independently. In LAN level, caches for separate LANs are managed independently. Discovery, delivery and dissemination are the three process involved in the communication of web caching [48]. The process of finding the location of the page to be cached is called cache discovery. Web caching protocols are used to discover the cached objects in cooperating proxy servers or cache servers. Caching architecture specifies the method of delivering cached pages by making direct connection between the cache containing the page and the client. Dissemination process deals with the method of delivering requested document to the proxy and served to the requested client [49]. Pull caching retrieves a document after a miss and makes a decision whether to cache it or not. Push caching is server initiated and documents are duplicated in several cache servers when the network traffic load is low. This is also known as mirroring [25]. Cache pollution means a cache contains objects that are not frequently used in the near future [50]. The phenomenon of remaining unpopular objects in the cache is called cold cache pollution. The objects that have been
previously popular may remain in the cache for some time even if they are not popular any longer. This phenomenon is called hot cache pollution.

Afonso et al. developed an approach for evaluating the web caching parameters such as response time, latency, CPU usage, memory and disk space, to be used in QOS characterization [51]. The measurement metrics that relate to web cache performance are throughput, latency, response time, connection capacity, cost, cache hit ratio and byte hit ratio. Throughput is the number of HTTP response per second or request per second or URL’s per second. Latency is divided into three components: internal, external and object creation latencies [19]. The internal latency consists of the time it takes the proxy server to transfer the object to the client. The external latency is the time it takes the server to respond and transfer the object to the proxy server. Object creation latency is the time taken by server to construct the dynamic object. Response time is the interval between the end of transmission of a request and beginning of the response.

Connection capacity is a metric used to measure how many simultaneous connection a cache can handle. Cost is a metric used to compare the communication cost, storage cost, implementation cost and lookup cost incurred due to caching/pre-fetching and analyze the performance of cheap caches with expensive caches. An ideal cache has an infinite size, stores all cacheable response and returns a hit whenever possible. Hit rate measures the effectiveness of the cache and the amount of bandwidth saved. Hit ratio is the ratio of locally available information to total volume of requests. Cache hits are the number of requested objects found in cache. Cache misses are the number of requested objects not found in cache. Cache hit ratio and byte hit ratio [14] are calculated
using equations 2.1 and 2.2. Percentage of cache hit ratio can be improved by sharing the cooperative caches, increasing cache capacity, maintaining consistency, making web pages into cacheable etc.

\[
\text{Cache hit ratio} = \frac{\text{Cache hits}}{\text{Cache hits} + \text{Cache Misses}} \quad (2.1)
\]

\[
\text{Byte hit ratio} = \frac{\text{Number of bytes served by cache}}{\text{Total bytes sent to clients}} \quad (2.2)
\]

Local hit means the requested document is available in local cache of proxy server, the client request can be satisfied from its cache. Local miss means the requested document is not available in local cache of proxy server, the client request cannot be satisfied from its cache. The client request is forwarded to the neighboring proxy servers or web server. Remote hit means the requested document is not available in local proxy server cache, but it is found in the neighboring proxy server’s cache, the client request can be satisfied from there and upon receipt of that document, it is cached locally in the requested proxy server. Remote miss means the requested document is not available in the local proxy server cache and neighboring proxy’s cache, this request is forwarded to the web server. The client request is satisfied from web server and upon receipt of that document, it is cached locally in the requested proxy server.

The cost and benefit synergies and tradeoffs in web caching are discussed in [4, 30, 32]. Tradeoffs between CPU cost, storage cost, cache size and network traffic are considered in a good caching system. Several benchmarking tools such as Wisconsin Proxy Benchmark, Blast, Web Polygraph, WebJamma are available for evaluating the performance of cache systems [40, 52, 53]. The useful tips for building a cache-aware site [54] are given in Table 2.1.
Table 2.1 Tips for Building a Cache Aware Site (from [54])

<table>
<thead>
<tr>
<th>Serial No.</th>
<th>Tips</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Use URL's consistently</td>
</tr>
<tr>
<td>2.</td>
<td>Use a common library of Images</td>
</tr>
<tr>
<td>3.</td>
<td>Make caches to store images and pages that don't change often</td>
</tr>
<tr>
<td>4.</td>
<td>Make caches to recognize regularly updated pages</td>
</tr>
<tr>
<td>5.</td>
<td>If a resource changes, change its name</td>
</tr>
<tr>
<td>6.</td>
<td>Don't change files unnecessary</td>
</tr>
<tr>
<td>7.</td>
<td>Use cookies only where necessary because cookies are difficult to cache</td>
</tr>
<tr>
<td>8.</td>
<td>Minimize use of Secure Socket Layer (SSL) because encrypted pages are not stored by shared caches</td>
</tr>
</tbody>
</table>

2.3 Benefits of Web Caching

Web caching reduces web traffic and alleviates network bottlenecks. Web caches have proved to be useful for latency reduction, bandwidth conservation and disconnected operation [4]. Proxy cache sharing further reduces the load on the original server and internetwork load. The benefits of web caching are given below.

- **Latency Reduction**

  In web caching, the proxy server first searches the document in local cache. If it is there in cache, quick response is possible with the required document. If the document is not in cache, it sends the request to the web server. Upon receipt of that document, proxy server caches it locally. Thus subsequent request for the
same document can be provided quickly from the cache and latency can be reduced.

- **Web Traffic Reduction**

  Since local caches satisfies most of the client requests by storing recently accessed documents in cache, traffic towards web servers are reduced. Reduction of web traffic results in internetwork load reduction and lower congestion.

- **Bandwidth Conservation**

  Whenever a cache avoids the transmission of bytes to or from the original web server, the bandwidth requirements on that portion of the network can be reduced. It improves the overall latency. Bandwidth conservation results in cost savings.

- **Disconnected Operation**

  Even if the original web server is unavailable, due to network disconnection or server failure, a proxy server can provide access to the information from the cache. But the subsequent hyper link that is not available in cache cannot be accessed from web servers. Only mirror sites can provide subsequent hyper links. This improves service to remote network resources, such as busy FTP sites and transient Gopher servers that are often unavailable remotely, but may be cached locally.

- **Reduces load on the Server**

  Since recently accessed objects are stored in cache, most of the client requests are satisfied from local cache. Number of client request that are to be transmitted through the network towards the origin web server are reduced. Thus the work load at the web server is reduced.
2.4 HTTP Support for Web Caching

The headers of an HTTP transaction can also specify aspects relevant to the cacheability of an object. Date, Last-Modified: date, If-Modified-Since, ETag, Expires, Pragma: no-cache, Cache-Control are the headers that are meaningful for caching. HTTP/1.1 support for web caching is described in [55]. Date header is used to specify the last time the object was considered to be fresh. If the HTTP GET request includes If-Modified-Since header, then web servers use the Last-Modified date on the current content to identify whether the object has changed after the date of the cached copy. Etag (entity tag) represents a signature of the object and it allows stronger test to verify the signature of the current object at this URL matches the cached one. Expires header is used for finding the expiry date of an object. Pragma:no-cache header appended to GET request can indicate that the object is to be reloaded from the server irrespective of whether it has been modified or not. Cache-Control header field is used to provide a general mechanism for caches to communicate efficiently and it provides the directives for regulating requests and responses along caching food chain [27]. Cache control directives are used for deciding whether a document is cacheable or not. Table 2.2 shows the meanings of cache control directives.

Table 2.2 Meanings of Cache Control Directives

<table>
<thead>
<tr>
<th>Cache Control Directives</th>
<th>Meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache-Control = “public”</td>
<td>Responses from this server may be</td>
</tr>
<tr>
<td>Cache-Control</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>&quot;private&quot;</td>
<td>Only private cache may cache the contents of this response</td>
</tr>
<tr>
<td>&quot;no-cache&quot;</td>
<td>Responses can be cached, but the cached copy may not be reused for subsequent requests without revalidating the cached copy with the originating server</td>
</tr>
<tr>
<td>&quot;no-store&quot;</td>
<td>Response may not be cached on nonvolatile storage</td>
</tr>
<tr>
<td>&quot;must revalidate&quot;</td>
<td>Caching service must revalidate the document after it becomes stale from the originating server</td>
</tr>
<tr>
<td>&quot;max-age=20000&quot;</td>
<td>Document contained in this response should be considered &quot;stale&quot; after 20000 seconds</td>
</tr>
</tbody>
</table>

### 2.5 Location of Cache Servers

Caching is performed in various locations throughout the web. The cache can operate on a client or server or on an intermediate system. Starting in the browser, a web request may travel through multiple caching systems on its way to the origin server [1]. At any point, if the request matches a valid response in the cache, a response may be satisfied from the cache. Browser cache at client side, forward proxy cache at organization and client’s Internet Service Provider (ISP), reverse proxy cache (gateway cache) at server's ISP or CDN and origin web server
side cache are the different types of caches in the World Wide Web. CDN is a network of servers that delivers content to users on behalf of content providers [56].

Browser cache is useful when user's hit "back" button or click a link to see a page they have just looked at. Browser caching has been implemented by most existing web browsers [57]. Proxy caches are shared by more clients. The web requests are directed to the proxy servers either by browsers or by the underlying network (Interception proxies). Gateway caches are also known as reverse proxy caches or surrogate caches or intermediaries. Gateway caches are deployed by web masters to make their sites more scalable, reliable and better performing. Content delivery networks distribute gateway caches throughout the Internet and sell caching to interested web sites. Examples of browser caches, proxy caches and gateway caches are given in [55]. Different possibilities for selecting the location of web cache servers [58] are given below.

1. Place web cache servers on the users side of a bottleneck
2. Place web cache servers where the flow of traffic begins
3. Place a local web cache server at the side of user's Internet connection
4. Place web cache servers where more networks join together

Depending upon the location of cache servers, caches can also be classified into institutional cache, first level cache, upper level cache etc. An organization's cache servers could be co-located with their web server or on a separate system such as proxy server. The first level cache server can be placed as close as possible to the external Internet connection or close to users. If several first-level network connections are available, place the first level cache server at the place where
networks come together. The upper-level cache servers can be placed where networks join together, on national Internet exchange point, on international Internet exchange point or close to these exchange points.

### 2.6 Web Caching Protocols

In web cache sharing, for each URL request, the proxy server first checks in its local cache. If there is no local hit, that proxy server checks in other cooperating proxy's cache for remote hit. If there is no remote hit, then it forwards that request to the web server. Upon receipt of that document, the requested proxy server stores it in cache and returns the document to the client. Push and pull are the two approaches for cache discovery [59]. In pull cache discovery, proxy tries to locate a cache containing requested web page and retrieves it. In push cache discovery, proxy tells other cooperating proxies about its cache contents and informs them about changes in its cache. Web caching protocols are used for finding remote hit in other cooperative proxy/cache servers. Various web caching protocols are discussed below.

#### 2.6.1 Internet Cache Protocol (ICP)

ICP is a web caching protocol used to exchange hints about the existence of URLs in neighbor caches. In ICP, if a cache miss occurs, that proxy sends query messages to other participating proxies to find remote hit [60, 61]. If one proxy has a cache miss, every other proxy has to receive and process a query message. As the number of collaborating proxy increases, the CPU and communication overhead also increases. ICP message format is described in [62]. Extended Internet Caching Protocol (x-ICP) contains the mobility extensions and the web
access pattern during periods of mobility [63]. In x-ICP, the proxy server of the nomadic user’s newly visited network can retrieve web objects from its home proxy server. Foreign proxy only contacts with the home proxy where a large number of objects may be downloaded by the same user. Using x-ICP, cache hit rate is higher compared to visit other cooperating proxy servers. Since there is no broadcasting or sending query messages to other cooperating proxies, the Internet traffic is reduced and response time is shortened. Waited-ICP (WICP) is used to improve the efficiency of cache cooperation for video streaming applications [64].

2.6.2 Content Routing Protocol (CRP)

CRP is used to locate cached content from the existing cache meshes. CRP uses multicasting to query cache meshes [40]. It eliminates the server bottleneck and places the data closer to the demand. Cisco provides content routing protocols for enterprises and Internet service providers to build content delivery network.

2.6.3 Cache Array Routing Protocol (CARP)

CARP is a hashing scheme. Microsoft uses CARP to identify the proxy which has the requested information [65, 66]. It is a distributed caching protocol based on a known membership list of loosely coupled proxies. The client has to find the hash value for the requested URL and send it to the cache server. The hashing function must be properly selected such that the pages are evenly distributed among cache servers. The cache routing scheme in CARP avoids the overhead and scalability issues related to the cache communication.
2.6.4 Web Cache Communication Protocol (WCCP)

Cisco uses WCCP for transparently redirecting the HTTP traffic from network routers to the cache engine [40]. WCCP provides routing of web traffic from the router to the web cache, automatic failure detection and load balancing among multiple web caches.

2.6.5 Caching Neighborhood Protocol (CNP)

CNP describes an infrastructure for building dynamic caching hierarchies. This scheme permits flexible communications among proxy servers and decreases the response time [67]. A caching neighborhood is a web content partial replication group whose members include one origin server and a variable number of proxy servers. CNP is a suite of protocols that govern operations of caching neighborhoods.

2.6.6 Two-Tier Cooperation Protocol (2TC)

In 2TC protocol, the cooperation among cache servers are achieved by Informed Protocol (INF) and Query Protocol (QP). 2TC embeds the two main cooperation policies for web caching systems. It uses informed cooperation approach among close cache servers and a query cooperation approach among distant cache servers. 2TC can explore the content of several cache servers through a single query message [68].
2.6.7 Cache to Master Protocol (CMP)

This protocol is used for supporting the Summary-Query scheme. CMP is based on ICP and it has the ability to report a hit in a third cache server not involved in the actual message exchange. CMP is lighter and simpler than the original ICP. Distributed cooperation models based on a two-tier organization of cache servers are proposed in [69] using CMP.

2.6.8 Cache Group Management Protocol (CGMP)

Web caches are organized into multiple and overlapping multicast groups which use voting and feedback techniques to estimate the admitting or excluding members from that group. CGMP specifies how the meshes are formed and how individual caches join and leave the meshes. The fundamental design goals of CGMP include making the entire cache topology group-wise connected, making the grouping follow and match the topological constraints and grouping caches to minimize the number of “hops” a URL must travel upon cache fault [70].

2.6.9 Hypertext Streaming Protocol (HTSP)

HTSP is a web page transmission protocol designed for controlling the transmission order of inline objects within a web page [71]. MGET method is introduced in the HTSP to retrieve multiple objects in the specified order by a single request. In the MGET request, the transmission order of inline objects is specified. Browser first retrieves HTML document in which transmission order description is specified. Browser next issues MGET request with transmission
scenario translated from the transmission order description. HTSP allows a client to request multiple objects by a single request. HTSP is extended for transmission order control via a caching proxy. The proxy retrieves the objects which are not in the cache from the server. Then proxy performs transmission serialization and transmits the serialized body to the client. The proxy may change the transmission order specified by a client and the client may receive the objects in the order changed by the proxy.

2.6.10 Directory Server Protocol

In this protocol, a server keeps track of cached documents of all proxies. The server serves queries for cache hits in other proxies. If the server is down, it could not satisfy the clients request and causes cache misses. If cache miss occurs, it has to forward that request to the origin web server [72].

2.6.11 Full Informed Protocol (FIP)

The neighbour proxies continuously inform each other about the locally cached documents. Each proxy maintains a directory of documents which are cached in all neighbour proxies. Whenever the content of the cache directory of a proxy changes, a notification message is send to all neighbour proxies [72]. Since global state information is available in each proxy, there is no need of discovery phase and false hits and false misses can be avoided. Since the local cache of each proxy changes frequently, FIP causes a large traffic of notification messages.
2.6.12 Cache Digest Exchange Protocol

A summary of the contents of the cache server (Digest) is fetched by other servers which peer with it. It allows cache servers to make information about their cached content available to peers for avoiding the query/response delay. Cache digests have a protocol and a data format for fetching summaries form a proxy server and storing the digests. It uses HTTP to transfer directory information. The keys which are looked up in the cache digests are formed by performing MD5 digest function on the concatenation of HTTP method and the requested URL. In order to add a public key in a cache digest, the following steps are calculated [73].

1. Calculate the MD5 signature from HTTP method and URL.
2. Split the resulting 128 bit value into N chunks.
3. For each of these N chunks, the corresponding index into the cache digest is the digest value modulo the number of bits in the digest.

2.6.13 Summary Cache

It is a scalable wide-area web cache sharing protocol. In summary cache scheme, each proxy stores a compact summary of the cache directory of every other proxy [46]. When a local cache miss occurs, the local proxy first probes all the summaries to see if the request might be a cache hit in other proxies, and sends a query message to fetch the document, only to the proxy whose summary shows the promising result. Three kinds of errors that can occur in this summary cache method are false misses, false hits and stale hits. The network overhead is determined by the frequency of summary updates, the number of false hits and
remote hits. The memory requirement is determined by the size of individual summaries and the number of cooperating proxies. The errors affect the total cache hit ratio or the inter proxy traffic. But they do not affect the correctness of the caching scheme and the errors are tolerable. Thus Summary cache is scalable.

The summaries need to be stored in the main memory because memory lookups are much faster and disk arms make bottlenecks in proxy caches. Since the memory requirement grows linearly with the number of proxies, it is important to keep the summaries as small as possible. The impact of update delays and summary representations are discussed in [46]. The summaries can be represented by any one of the following methods.

(i) **Exact directory approach:** Summary contains exact URL's in the cache directory and each URL is represented by 16 bytes MD5 signature. This method requires more memory for storing the summaries.

(ii) **Server name approach:** The summary is the list of server name component of the URL's in cache. The server name approach reduces the memory needed for storing summaries. The URL requested might have the same server name with different filenames. So there is a chance of increasing false hit.

(iii) **Bloom filter approach:** In bloom filter approach, using hash functions, proxy builds a bloom filter from the list of URL's of cached documents and sends a bloom filter bit array, with the specification of the hash functions to the other proxies. A bloom filter is a method for representing a set $S = \{s_1, s_2, ..., s_n\}$ of $n$ elements to support for membership queries. Initially, $m$ bits of vector $v$ are set to $0$ and $k$ independent hash functions $h_1, h_2, ..., h_k$, are selected. For each
element $s \in S$, the bits at positions $h_1(s)$, $h_2(s)$, ..., $h_k(s)$ in vector $v$ are set to 1. A particular bit might be set to 1 multiple times. When a query for $q$ is given, the bits at positions $h_1(q)$, $h_2(q)$, ..., $h_k(q)$ are checked. If any of them is 0, then certainly $q$ is not in the set $S$. If all elements at the corresponding bit positions are 1, then $q$ may be there in the set $S$. There is certain probability that $q$ may not be there in the set $S$. This is called "false positive" or "false drop". This false positive causes false hit to occur. The parameters $k$ and $m$ should be chosen such that the probability of a false positive is acceptable. As the size of the bloom vector increases the chance of getting false positive decreases. The advantage of bloom filter is that they need only less memory for storing summaries. As the number of URL's in the cache directory increases, there is a chance of increasing false positive.

(iv) **Abbreviation method:** Abbreviation method is proposed in [74], in which each URL in summary is represented by 4 bytes. The summary is formed by concatenating the first character of the server name, first character of the first directory name, first character of the filename and the first character of the extension name of the filename. In this method summary size is only 4 bytes per URL and the chances of getting false hits are less. Abbreviation method can be modified further for providing better remote hit ratio by slight increase in summary size. The size of summary can be extended by combining first character of server name, first characters of directories and subdirectories, first character of filename and first character of its extension name. Then the chance of getting duplicate abbreviation can be reduced. The advantage of abbreviation method is that they provide a tradeoff between the memory requirement and the false hit.
2.7 Web Caching Architectures/Techniques

Cache collaboration is important to improve caching performance. When one of the clients sharing the proxy generates a request, the proxy searches its local storage for the requested object. If the object is available locally it is sent to the client otherwise the request is passed on to the remote cooperative proxy servers or cache servers. If there is no remote hit, the client requests are forwarded to the origin server. Web caching techniques can be characterized as back-end or front-end [75]. Back-end approaches are deployed inside the site infrastructure at the original server side. They can reduce content generation delays but they do not address network related delays and bandwidth consumption issues. Front-end approaches concern caching outside the site infrastructure at a proxy side or a cache that resides at the edge of a Content Delivery Network (CDN). Front-end web caching techniques are serving at client side and they have difficulty in handling dynamic web pages due to their strong dependency on the back-end site infrastructure.

In proxy caching, proxy server intercepts HTTP requests from clients and if it finds the requested object in its cache, it returns the object to the client. If the object is not found in proxy's cache, it forwards the request to the original server, gets the object, deposits it in cache and returns the object to the user. One disadvantage of this design is that the cache represents single point failure in the network [76]. This drawback can be avoided by sharing the caches of proxy servers [77]. In reverse proxy caching, caches are deployed near the servers. This mechanism is useful for servers that expect a high number of requests and want to assure a high level of quality of service [40]. In application-level caching,
caches are explicitly managed by application. It is designed with an API which allows an application writer to explicitly manage the cache contents [78]. The information about proxy caches that is used in filtering the request is called hint information. Hint allows clients to make decisions based on local state [79]. A shared cache solution for the home Internet gateway is outlined in [80]. For avoiding proxy overhead, site-based dispatching (SBD) technique forwards only hot-site requests to the proxy. SBD forwards request to less popular sites directly to the remote web sites [81]. Various web caching architectures/techniques are discussed below.

2.7.1 Cooperative Caching

In cooperative caching, caches communicate with peers over the network before demanding files from the web as shown in Figure 2.3. Cooperative proxy caching is the sharing and coordination of caches among multiple caching proxies [82, 83, 84, 85]. Cooperation among caches can be performed in horizontal and vertical orthogonal dimensions. Horizontal cooperation is performed by geographically clustered caches that have equal distant to the web servers. Vertical cooperation is performed by geographically distributed caches that have unequal distant to the web servers.

An expiration age-based scheme is proposed in [86] for reducing the replication of documents across the cache group, while ensuring that a copy of the document always resides in a cache where it is likely to stay for the longest time. In buddyCache approach, a group of close-by collaborating clients are connected to a storage repository. They can avoid interactions with the server if needed
objects and coherency information are available in any client in the group [87]. Cooperative caching architectures can be divided into hierarchical, distributed and hybrid caching architectures [88].

2.7.1.1 Distributed Caching

Distributed cache is a set of cooperative caches placed at the same level of the network and missing of resources at one proxy causes a search in all cooperating cache servers for cache hit. In distributed caching, no intermediate caches are set up and there are only institutional caches at the edge of the network which cooperate to serve each other's misses [89, 90, 91]. Figure 2.4 shows how proxy servers are connected to each other in distributed caching architecture. Each proxy server can communicate to each other and it can send query messages or

Figure 2.3 Cooperative Caching (Redrawn from [1])
keep digest/summary to find remote hits. Distributed caching architecture is developed for improving hit ratio, providing load balancing and reducing web traffic. Distributed caching architecture with centralized control is proposed in [92]. In this architecture, a set of cooperative caches deployed in a network as an aggregate pool such that if a web object is stored in any one of the caches, it can be retrieved by a client request. Sharing of retrieved documents among browsers using Distributed Shared Memory (DSM) model is described in [93]. A directory structure is developed in [94] for allowing individual cache servers to locate objects cached at neighbouring sites, combining them into a logically unified collective cache.

![Figure 2.4 Distributed Caching](image-url)
Design goals and guidelines for designing distributed caches are discussed in [95]. Design goals of a distributed web cache should include low discovery cost, object dissemination that adapts to rapid shifts in popularity, a method for selecting fast cache sites and concurrent delivery of page or object components from different caching sites. The design principles of large scale distributed caches [96] are given below.

- Minimize the number of hops to locate and access data on both hits and misses
- Share data among many users and scale to many caches
- Cache data close to clients

**Advantages:**

Each proxy can communicate to each other and it can send query messages or keep digest/summary to find remote hits in the same level of the network. So most of the traffic flows through the less congested lower network levels. Distributed caching has lower transmission time than hierarchical caching. In distributed caching, cache hit ratio is increased and latency is reduced.

**Disadvantages:**

In distributed caching, local network traffic is increased and memory requirement is increased for storing the summary/cache digest. As the number of proxies increased, CPU processing overhead and communication overhead are increased. False hits, False misses and stale hits can occur.
2.7.1.2 Hierarchical Caching

Proxies or cache servers are arranged in a tree like structure either logically or physically as in Figure 2.5. In hierarchical caching architecture, caches are placed at multiple levels of the network [97]. Individual caches can be interconnected hierarchically to mirror an inter network's topology [98]. Each bottom level cache is associated with a set of clients. A client request is first sent to the bottom level cache and then iteratively forwarded up the hierarchy such as institutional cache, regional cache, national cache, until the request is satisfied. If the root cache does not have the target object, the request is finally directed to the origin server [99].

Dynamic web caching hierarchies consist of proxy servers. Hierarchies are built based on the requests from clients. In dynamic caching hierarchies, a proxy server may be level one cache for one request and level two cache for another [67]. Disk requirements of institutional caches in distributed caching are much smaller than top-level caches of hierarchical caching.

Advantages:

Hierarchical caching has lower bandwidth usage than distributed caching. In hierarchical caching, cache hit ratio is increased and latency is reduced.

Disadvantages:

Distributed caching has higher transmission time than hierarchical caching. As the number of proxies increased, CPU processing overhead and communication overhead are increased. False hits, False misses and stale hits can also occur.
2.7.1.3 Hybrid Caching

Hybrid caching architecture combines hierarchical caching with distributed caching at every level of a caching hierarchy [88]. In a hybrid scheme, cache cooperates with other caches at the same level or at a higher level using distributed caching. The document is fetched from a parent/neighbor cache that has the lowest round trip time. ICP protocol can be used to find the remote hit in the parent/neighbor caches. Compared to distributed and hierarchical caching, this approach reduces the delay in accessing the documents from remote proxies [89].
Advantages:

In hybrid caching, document is fetched from a parent/neighbor cache that has lowest round trip time. Compared to distributed and hierarchical caching, this approach reduces the delay in accessing the documents from remote proxies.

Disadvantages:

In hierarchical caching, local network traffic is increased and memory requirement is also increased for storing the summary/cache digest. As the number of proxies increased, CPU processing overhead and communication overhead are increased. False hits, False misses and stale hits can also occur.

2.7.2 Transparent Web Caching

Transparent web caching uses network devices to redirect HTTP traffic to cache servers. This technique is called transparent because Web browsers do not have to be explicitly configured to point to a cache server [101]. There are two ways to deploy transparent proxy caching: at the switch level and at the router level. Router based transparent caching uses policy-based routing to direct requests to the appropriate caches. In switch based transparent caching, the switch acts as a dedicated load balancer [76]. The clients and cache servers are connected to the Layer 5 switch as in Figure 2.6.

Layer 5 switches use information in the TCP and HTTP request headers. This improves performance by ensuring that non-cacheable HTTP requests bypass the cache servers. LB-L5 Web caching scheme uses the Layer 5 switching based technique to support distributed web caching [102]. LB-L5 uses a weighted bloom filter to represent cache content and access frequency information. LB-L5 extends
ICP to support communication between cache servers and L5 switches [103]. In this method, when the number of URL's in the cache directory increases, there is a chance of increasing false positive. Thus remote hit ratio is reduced.

Advantages:

In transparent caching, dynamic requests can be directly forwarded to the web server. Load balancing among the cache servers can be achieved. Deciding of cache server to fetch the document is done faster.

Disadvantages:

When the number of URL's in the cache directory increases, there is a chance of increasing false positive. Thus remote hit ratio is reduced.

2.7.3 En-route Caching

In this architecture, the caches are associated with routing nodes in the network. These routing nodes are called en-route caches [104]. An en-route cache intercepts all client requests passing through the associated routing node as seen...
in Figure 2.7. If the requested object is in the cache, the object is sent to the client and the request is not propagated further upstream. Otherwise, the routing node forwards the request along the regular routing path towards the origin server. If no en-route cache has the target object, the request is eventually serviced by the origin server. Since no request is detoured off the regular routing path, the additional bandwidth consumption and network delay for cache misses are minimized. The extra overhead needed for locating the cache server by sending broadcast queries or maintaining cache summaries are eliminated in en-route caching.

In en-route caching, caches are often located throughout the global network instead of being deployed within the user organization only. So vertical cooperation is more commonly used in en-route caching. The performance of en-route web caching depends on the locations of the caches and how the cache contents are managed. Dynamically determining the appropriate number of object copies and placing them in suitable en-route caches are challenging tasks. Xueyan et al. developed a scheme that dynamically places object in the caches on the path from the server to the client in a coordinated fashion [104]. They have obtained the optimal locations for caching objects using a dynamic programming algorithm. The cache location problem for en-route caching is discussed in [105, 106].

**Advantages:**

In en-route caching, caches are transparent to both origin servers and clients. The overhead of sending broadcast queries or maintaining cache summaries are eliminated in en-route caching.

**Disadvantage:** In en-route caching, stale hits can occur.
2.7.4 Cascaded Caching

Cascaded caching is also known as multi-layer caching. Multiple copies of same object may be available in many caches placed at different locations. There can be cache servers both at the client location and server location. This allows a cache server at client location to get a requested document from the cache server at the server location. The overall performance of cascaded caching depends on how the cache contents are managed, including object placement and replacement algorithms [100]. Figure 2.8 shows that normally more than one
cache is located between the origin servers and the clients. Dynamically determining the appropriate number of object copies and placing them in suitable caches are challenging tasks.

Figure 2.8 Cascaded caching (Redrawn from [100])

**Advantage:**

URL's can be processed by the content server aliases, so that requests from firewall proxies can be forwarded to their destinations as directly as possible.

**Disadvantages:**

It is difficult to determine the appropriate number of object copies and placing them in suitable caches. Allocating too many copies of unpopular objects in the network is wasteful of cache space while assigning too few copies of popular objects may not reduce their access latency sufficiently. Stale hits can also occur.
2.7.5 Multicast Web Caching

Multicast web caching is a technique to receive HTTP content via an IP multicast group and to store a selection of all received content in a local cache. In order to reduce the received traffic, only popular objects are transferred via multicast and unicasting is used for all others. A server proxy cache deployed at the central access point communicates over the broadcast network to the client proxy caches located at the client side using IP multicast [107]. Each client node is able to handle its own traffic and the traffic generated by all the other clients. Each client node requests an object via unicast and the server decides whether it should be multicasted. Each client node can decide whether to receive the multicasted objects or not. It can also switch between hierarchical caching and multi caching. In popularity based layered multicast web caching, m multicast channels are used. Most popular documents are transferred through the first layer and least popular ones are transferred through the $m^{th}$ layer. One of the available channels is selected for the transfer of a requested document based on its popularity. The server or the client proxy can select the channel to be used for multicast transfer. This scheme allows significant reductions in the required processing and storage capacity while introducing a little inefficiency.

**Advantages:**

CPU processing overhead, storage overhead and communication overhead are reduced significantly.

**Disadvantages:**

It introduces little inefficiency in identifying the popular objects. Stale hit can also occur.
2.7.6 Adaptive Caching

An adaptive web caching system has a scalable, robust, adaptive and fully distributed protocol for self-organizing cache servers into overlapping multicast groups [108]. Web caches maintain a URL routing table. The primary keys of URL routing table are URL prefixes, associated with one or more identifiers to the next hop caches or cache groups. The routing table is used for deciding whether to forward a request to another cache in the web caching infrastructure. Adaptive algorithms are used for exchanging of information among caches in a cache mesh. Cache group management protocol is used for making the entire cache topology group-wise connected and minimizing the number of hops a URL request must travel upon cache fault [70]. Adaptive distributed caching (ADC) algorithm combines the advantages of hierarchical distributed caching (allowing multiple copies of the same object) and hashing based distributed caching (fast allocation through global agreement) [109].

Advantages:

Adaptive caching minimizes the number of hops a URL request must travel upon cache fault. It combines the advantages of hierarchical distributed caching and hashing based distributed caching.

Disadvantages:

In adaptive caching, stale hit can occur. As the number of cache servers increase, the URL routing table size grows and processing overhead can occur.

2.7.7 Value-Based Web Caching

In this web caching technique, redundant data transfers due to resource modification and aliasing are eliminated by caching data based on its value,
rather than its name. Aliasing occurs when the same data is named by multiple
URIs. In value-based web caching (VBWC) algorithm, data is broken into blocks
and each block is named by its image under a secure hash function such as MD5
[110]. This image is called the block's digest. Another table is also used for
mapping resources to the blocks of which they are composed. Indexing of web
resource data result in better utilization of storage resources.

Advantages:

Redundant data transfers due to resource modification and aliasing are
eliminated. When aliasing occurs, the aliased data is stored only once.

Disadvantages:

In value-based caching, stale hit can occur. As the number of cached document
increases, CPU processing overhead and storage overhead are increased.

2.7.8 Peer-to-Peer (P2P) Caching

Peer-to-peer systems capitalize on individual user resources for building
scalable and self organizing file sharing systems. P2P systems are classified into
two classes: pure P2P where peers share data without a centralized coordination
and a hybrid P2P where some operations are intentionally centralized [111].
Gnutella protocol specifies the operations for maintaining and searching a peer­
peer overlay network [112]. Squirrel is a cooperative P2P web cache that relies on
pastry protocol [113]. Client requests are forwarded to the squirrel proxy running
on client’s machine. If the object is uncacheable then the proxy forwards the
request directly to the origin web server. Otherwise it checks the local cache, if a
fresh copy of the object is not found in this cache, then Squirrel tries to locate it on some other node.

The pastry protocol provides distributed hash table and routing functionalities. Pastry forwards the request to a participating node containing the requested object that is closest to the requested client. This node becomes the home node for accessing that object. Squirrel has two schemes for storing the copy of the cached objects. In home-store scheme, objects are stored both at client caches and at its home node. In directory scheme, the home node for an object maintains a small directory of pointers to nodes that have recently accessed objects. Request for these objects are sent randomly to one of these nodes. Stochastic fluid model for P2P caching evaluation was developed in [113], which uses an infinite state Markov process for decreasing the complexity of computing the hit probability.

**Advantages:**

In peer-peer caching, the complexity of computing the hit probability is reduced. Uncacheable requests are directly forwarded to the web server.

**Disadvantages:**

In peer-peer caching, stale hit can occur. As the number of participating nodes increased, CPU processing overhead and storage overhead are also increased.

**2.7.9 Server Side Caching**

Even if client side caches could not satisfy the user requests, a cache at sever side is useful in reducing the delay in accessing the documents from the server’s hard disk. Temporal and geographical localities of reference are exploited on a
much large scale at server side [114]. Web server accelerator contains a cache and load balancer. It resides in front of a web server for delivering cached responses and leaving the role of content generation to the web server. Cached objects are directly sent from the accelerator to the clients.

A high performance web server accelerator called Tornader was developed in [115] for optimizing the web content delivery and improving the performance of web servers. Their experimental results show that Tornader boosts the performance of the Apache web server up to 150% in both single and dual CPU cases. Design and performance of a scalable web server accelerator is discussed in [116]. Architecture of a web server accelerator is shown in Figure 2.9. Multiple web servers would be needed at a web site which receives a high volume of requests. TCP router examines the contents of a web request and a content router routes it based on the URL requested [117]. Neural network modeling is proposed
in [118] to map web page requests to web server caches for maximizing hit ratio while load balancing among caches.

**Advantages:**

It boosts the performance of the web servers. Cached objects are directly sent from the accelerator to the clients. It maximizes the cache hit ratio and provides load balancing among caches.

**Disadvantages:**

If the number of cached documents is increased then the storage overhead is increased.

2.7.10 Caching of Dynamic Contents

Dynamic web pages are created on request by application programs stored in the back-end site infrastructure. Caching of dynamic web pages are essential for improving the performance of web sites containing significant dynamic content and information personalized to individual users [119]. An examination of several dynamic web pages on the web revealed that only parts of the pages are dynamic in nature. Dynamic content has three forms of locality: identical requests, equivalent requests and partially equivalent requests [120]. Identical requests have identical URLs which result in the generation of the same content. The URLs of equivalent requests are syntactically different but result in generation of identical content. Partially equivalent requests are syntactically different but results in generation of content which can be used as temporary place-holder for each other.
Web pages can be divided into fragments for providing content generation and caching. A fragment is a portion of a web page which has a distinct functionality and is distinguishable from the other parts of the page. These fragments are cost effective independent units and can be stored in cache. Manual fragmentation of web pages are expensive, error prone and unscalable. An automatic fragment detection method is proposed in [121] for the web sites serving dynamic content. WebGraph framework is proposed in [122] for improving the response time for accessing dynamic objects. It uses a graphical representation of web pages to serve dynamic web pages efficiently. Dynamic web pages are divided into multiple components known as weblets. The weblets can be static or dynamic and are used to construct and reconstruct the web pages. Web pages can be formed from these weblets by creating templates in a markup language. Upon access to a dynamic page, instead of recreation of the entire page, only the weblets that have changed are recreated and integrated with the precomposed page. WebGraph framework facilitates web caching, quality of service support, load balancing, overload control and security for both dynamic and static web pages.

In DOMProxy (Dynamic Objects Manager Proxy) Architecture, front-end web caching is used for dynamically generated web pages [75]. Even though whole dynamically constructed HTML documents are not suitable for caching, HTML documents constructed using constant templates can be cached [123]. Database caching at proxy servers enables dynamic content to be generated at the edge of the network for improving the scalability and response time of web applications [124, 125]. Cache clouds architecture is proposed in [126] for cooperative caching of dynamic documents in edge networks. A cache cloud contains caches of an edge network that are located in close network proximity. A dynamic hashing
based cooperation scheme is used for efficient document lookups and document updates within each cache cloud. The beacon point of a document maintains the up-to-date lookup information containing the list of caches in the cloud that currently hold the document. If the server wants to update a document, it sends an update message to its beacon point. A random hash function is used for assigning beacon points for documents.

**Advantages:**

It increases cache hit ratio and reduces latency. It makes provision to cache more web pages.

**Disadvantages:**

Manual fragmentation of web pages are expensive, error prone and unscalable. It is difficult to determine what pages should be cached when a cached page becomes obsolete.

**2.7.11 Active Caching**

Active caching supports caching of dynamic contents at proxy servers. The web server supplies cache applets along with the attached document and the proxy servers cache these applets. Applets in the proxy are invoked upon cache hits to furnish the necessary processing without contacting the server. This scheme migrate the parts of server processing on each user request to the caching proxy via 'cache applets'. The proxy has the freedom to manage its own resource and the association between cache applets and URLs. This scheme automatically migrate the server processing to nodes that are close to users. The active cache protocol, active cache interfaces and security mechanisms are discussed in [127].
**Advantages:**

In active caching, the parts of server processing is migrated to the caching proxy. The proxy has the freedom to manage its resources and it increases the scalability of web based services.

**Disadvantage:**

In active caching, security mechanisms are to be provided in the proxy server.

2.7.12 Soft Caching

Soft caching is a web cache management technique for images. In soft caching framework, cache management task determines which image to maintain in the cache and level of resolution at which they should be stored [128, 129]. Objective of the cache management algorithm is to minimize the average access time per image. For each URL request, the user is served with images from cache. That means whatever resolution available in cache is served first. If the user wants to get full resolution image, it can be fetched by pressing “reload”.

**Advantage:** Soft caching minimizes the average access time per image.

**Disadvantages:**

Users are getting whatever resolution of images available in cache first. Stale hits can also occur

2.7.13 Caching of Streaming Media

Media streams such as audio and video streams can be accessed or cached in portions. They have real-time delivery constraints. Caching of streaming media
objects are attractive due to their static nature, long duration and predictable sequential nature of access and high network resource requirements. In passive caching, proxy intercepts data sent from server to the clients and stores it for future use [32]. In pre-fetch caching, proxy proactively requests data from the server during non-peak times and caches the data for future use. For caching of media streams, the caching proxy needs to estimate the number of users who will be requesting each portion of media stream and proxy has to decide whether to cache these portions of media streams. An algorithm is devised in [32] for optimally selecting data units to cache based on the cost and distortion characteristics of the data units.

In prefix caching of streaming media, first few portions of the stream are cached first. When a user requests a stream, the caching proxy transmits the stored prefix of the requested stream and simultaneously requests the remainder of the stream from the server. This method reduces the starvation and buffer underflow. In video staging, the parts of variable bit rate stream where the bit rates are higher than the nominal bandwidth of server-proxy-user link are cached and the bandwidth required is smoothened [32]. In segmented caching of streaming media, blocks of a media object received by the proxy server are grouped into variable-sized distance-sensitive segments [130]. The initial portion of the stream is given importance and cached it. The number of segments is dynamically determined by the cache admission and replacement policies. In network-aware partial caching, popularity of streaming media objects, bit-rate requirements and available bandwidth between clients and servers are taken into account for caching of streaming media [131].
Advantages:

Caching of streaming media increases cache hit ratio, reduces latency and network resource requirements. It permits the real time delivery of data.

Disadvantages:

Caching of streaming media can cause buffer underflow at the receiving side. Stale hits can also occur.

2.8 Cache Replacement Strategies

When the cache becomes full and old, objects must be removed to make space for new ones. Replacement policies decide which data is to reside in cache. The goal of the replacement policy is to make the best use of available resources such as network bandwidth, disk and memory space [132]. An efficient replacement algorithm is to be used for minimizing the service cost in the network of caches. The replacement strategies can be classified as recency based strategies, frequency based strategies, recency/frequency based strategies, function based strategies and randomized strategies. The advantages and disadvantages of all these strategies are discussed in [133]. Efficient replacements of nonuniform objects in web caches are outlined in [134, 135]. Design, implementation and performance of proxy cache replacement algorithms are discussed in [136]. Vakali shows that evolutionary techniques are more beneficial for cache replacement compared to the conventional replacement techniques [137]. Cache replacement policies of p2p traffic are discussed in [138]. Adaptive replacement, coordinated replacement, combination of cache replacement and cache coherence,
multimedia cache replacement and differentiated cache replacement are new areas that need further investigation [133].

2.8.1 Recency-based Strategies

Strategies that incorporate size and/or cost into the replacement process are treated in the recency-based strategies class. It exploits the temporal locality seen in web request streams. LRU, LRU-Threshold, Pitkow/Reckers strategy, SIZE strategy, LRU-Min, EXP1, Value-Aging, Pyramidal Selection Scheme, Partitioned Caching are some of the recency-based strategies [133]. Recency-based strategies are better than other strategies because web request streams usually exhibit temporal locality and they are simple to implement and fast. Most commonly used recency-based strategy is Least Recently Used strategy. Simple LRU strategies do not combine recency and size factors. They do not consider frequency information for replacing objects in cache. Various recency based strategies are given below.

- **LRU**: This strategy removes the least recently referenced object to make space for new objects entering into the cache. It does not consider variable size and cost of objects [139]. Since implementation of LRU is simple, it is widely used in web caching.

- **LRU-Threshold strategy**: This strategy works like LRU and an object is not cached when the size of the object exceeds a given threshold.

- **Pitkow/Reckers strategy**: This strategy uses LRU and objects that are requested on the same day are differentiated by their size and the largest files are removed first.
• **SIZE**: In SIZE strategy, LRU strategy is applied to objects with the same size and the biggest objects are removed first.

• **LRU-Min**: This strategy tries to minimize the number of documents replaced. LRU-Min algorithm is given below.

1. Set the size of the requested document as threshold.
2. Prepare a list containing documents whose size is equal to or larger than threshold.
3. Apply LRU to that list until the list is empty or the free cache space is at least the threshold.
4. If the free cache space is not at least the size of the requested document.
   Set threshold = threshold/2 and go to step 2.

• **EXP1**: This strategy is a variant of LRU and the time period between the current time and the last request time are used to weight the importance of an object for replacement.

• **Value-Aging**: This strategy uses a formula for updating the value of an object and objects having larger values are removed first. The new value \(V_{\text{new}}\) of an object \(i\) is calculated from old value \(V_{\text{old}}\), current time \(C_t\) and last request time \(t_i\) of object \(i\) as given in equation 2.3.

\[
V_{\text{new}}(i) = V_{\text{old}}(i) + C_t \times \frac{\sqrt{(C_t - t_i)}}{2}
\]  

(2.3)

• **Pyramidal Selection Scheme (PSS)**: This strategy makes a pyramidal classification of objects depending upon their size. Each class has a separate LRU list. For replacement, the values of the least recently used object in each class are compared and choose the largest one from each class.

• **Partitioned Caching**: This strategy classifies the web objects into three groups (small, medium, large) according to their size. The thresholds for this
classification are derived from prior web traces. Each group is managed independently and has its own cache space. The following relation should hold while assigning cache size for each group.

Cache Size of Small group < Cache Size of Medium group < Cache Size of Large group

2.8.2 Frequency-based Strategies

Main factor of frequency-based strategies are the use of frequency information for finding the less frequently accessed objects. In these strategies less frequently used objects are removed first. Different replacement strategies available based on frequency are Less Frequently Used (LFU), Perfect LFU, In-Cache LFU, LFU-Aging, α-Aging and swLFU [133]. Some strategy uses popularity values of each object as their frequency value. The popularity of objects does not change very much over a specific time is the advantage. LFU-based strategies require more complex cache management, many objects may have similar frequency count and cache pollution are the difficulties faced in these strategies. Various frequency based strategies are given below.

- **Less Frequently Used**: The popularity values of web objects are used as frequency values. These frequency values are used for deciding the objects for replacement.

- **Perfect LFU**: Perfect LFU counts all requests to an object. Request count is to be kept for all objects seen in the past. This count is used for deciding the objects for replacement. In this strategy, space overhead is more.

- **In-Cache LFU**: In this strategy, counts are defined for cache objects only. Management is easy and space overhead is less.
• **LFU-Aging**: Popularity values of some objects used in a particular period may be larger and remain in cache even if they are not requested for a long time period. An aging factor is used for avoiding this cache pollution. In this strategy, if the average value of all frequency counters exceeds a threshold, all frequency counters are divided by 2. Objects with less frequency values are removed first.

• **α-Aging**: A periodic function is used for calculating explicit aging. Virtual clock is used for finding the age of each object. The age of every object is decreased for each virtual clock tick and each cache hit causes to increase the age based on a periodic function.

• **swLFU**: In this strategy, server can influence the caching of objects. A weighted frequency counter is used for each object for indicating the server appreciation about caching that objects. Objects with less frequency values are removed first to make space for new ones. The LRU strategy is applied to objects with the same weighted frequency value for breaking the tie.

### 2.8.3 Recency/Frequency-based Strategies

These strategies use recency and frequency as the main factor for finding an object for replacement. Segmented LRU, Generational Replacement, HYPER-G and LRU-SP are some of the strategies available under this category [133]. These strategies introduce additional complexity due to special procedures required for cache replacement. Various recency/frequency based strategies are given below.

• **Segmented LRU (SLRU)**: This strategy partitions the cache into two segments: an unprotected segment and a protected segment. The objects are inserted into the unprotected segment on the first request for an object. The
object is moved to the protected segment on a cache hit. LRU strategy is used for managing both segments.

- **Generational Replacement**: In this strategy, n lists are used for storing all objects. If an object is requested i times, that object is inserted into the ith list. List n contains all objects with n or more requests. A request to an object causes its deletion in the current list and insertion in the beginning of next list. Replacement takes place only at the end of list 1.

- **HYPER-G**: LRU, LFU and SIZE strategies are combined in this strategy. The least frequently used objects are chosen first for replacement. If more than one object meets this criterion, least recently used object is chosen from the cache. If again more objects meet the same criterion, the largest object is chosen for replacement.

- **LRU-SP**: This strategy is an extension of pyramidal selection scheme. LRU-SP strategy maintains different classes of LRU list. For each object, its class is determined by \( \log(s_i/f_i) \), where \( s_i \) is the size of object i and \( f_i \) is the number of past requests to object i. Object’s class may be changed for each request to that object. The values of least recently used objects in each class are compared for replacement. LRU-SP chooses object i for replacement only if its value \( (\Delta T_i s_i)/f_i \) is the largest among all other values of objects, where \( \Delta T_i \) is the number of accesses since the last time object i was requested.

### 2.8.4 Function-based Strategies

A general function is used in these strategies for finding an object for replacement. Greedy Dual size (GD), GDSF and Taylor Series Prediction (TSP) are some of the strategies available under this category [133]. They do not
consider a fixed combination of factors or fixed usage of data structures. The performance metric can be optimized by proper choice of weighting parameters. Many factors are considered in these strategies for handling different workload situations. But in these strategies, choosing appropriate weight is a difficult task. Using latency in the value calculation can lead to inferior replacement decisions. Various function based strategies are given below.

- **Greedy Dual Size**: A characteristic value $H_i$ is maintained for each object. $H_i$ is recalculated for request of each object. $H_i$ is calculated as equation 2.4.

$$H_i = \frac{c_i}{s_i} + L$$  \hspace{1cm} (2.4)

where $L$ is a running aging factor which is initialized to zero, $s_i$ is the size of object $i$ and $c_i$ is the cost to fetch object $i$ from its origin server. In GD-Size, object with smallest $H_i$ value is used for replacement and this value is assigned to $L$.

- **GDSF**: This strategy is similar to GD-Size and $H_i$ is calculated as equation 2.5.

$$H_i = f_i \times \left( \frac{c_i}{s_i} \right) + L$$  \hspace{1cm} (2.5)

where $f_i$ is the number of past requests to object $i$, $s_i$ is the size of object $i$ and $c_i$ is the cost to fetch object $i$ from its origin server. Objects with smallest $H_i$ values are used for replacement for making space for new ones.

- **Taylor Series Prediction**: This strategy is similar to GD-Size and $H_i$ is calculated as in equation 2.6.

$$H_i = \frac{(f_i \times c_i)}{(s_i \times T_T)}$$  \hspace{1cm} (2.6)

where $T_T$ describes the temporal acceleration of requests to object $i$, $s_i$ is the size of object $i$ and $c_i$ is the cost to fetch object $i$ from its origin server. $T_T = T_p - C_t$, where $T_p$ is the predicted time for the next request and $C_t$ is the current time. Second order Taylor series is used to find out $T_p$, it uses the last and the next to last request times. Objects with smallest $H_i$ values are removed first.
2.8.5 Randomized-based Strategies

Randomized decisions are used in these strategies to find an object for replacement. The different strategies proposed under this category are RAND, HARMONIC, LRU-C, LRU-S and Randomized replacement with general value functions [133]. These strategies do not need special data structures for inserting and deleting objects. They are simple to implement. It is difficult to evaluate randomized strategies. Different simulation run on the same web request trace will give slightly different results. Various randomized-based strategies are given below.

- **RAND**: This strategy removes a random object. This strategy gives equal probability to each object.

- **HARMONIC**: In this strategy, one item is selected at random from cache for replacement, with a probability inversely proportional to its specific cost as in equation 2.7.
  \[
  \text{Cost} = \frac{C_i}{S_i} \quad (2.7)
  \]
  where \(C_i\) is the cost to fetch object \(i\) from origin server and \(S_i\) is the size of object \(i\).

- **LRU-C and LRU-S**: These strategies are randomized versions of LRU. \(C_{\max}\) is the maximum access costs of all \(n\) objects in a request sequence as given in equation 2.8. \(\hat{C}_i\) is the normalized cost for object \(i\) as given in equation 2.9. The object is moved to the head of the cache with probability \(\hat{C}_i\) when an object is requested. Objects at tail of the cache are removed first for making space for new ones. LRU-S strategy is similar to LRU-C, it uses size instead of cost.
  \[
  C_{\max} = \max\{C_1, C_2, \ldots, C_n\} \quad (2.8)
  \]
  \[
  \hat{C}_i = \frac{C_i}{C_{\max}} \quad (2.9)
  \]
Randomized replacement with general value functions: This strategy takes \( n \) objects from the cache and evicts the least useful object in the sample. After replacing the least useful object, the next useful objects are retained in memory. At next replacement, new samples drawn and the previously retained least useful objects are evicted. The usefulness of a document can be determined by any utility function.

2.9 Cache Consistency Mechanisms

Cache consistency ensures that the cached object does not reflect stale or defunct data. Stale or defunct data refers to locally cached objects which are either no longer equivalent to the object on the originating server or obsolete. Web pages stored in cache must update frequently so that the cache can provide fresh copy as possible. Frequent update of cached copies can increase network traffic. HTTP protocol has provided some mechanisms for maintaining cache consistency [140]. Each URL has a “time-to-live” or “expire” field which is a priori estimate of how long the document will remain unchanged. Cache manager uses this field to determine whether the cached copy is up-to-date. Each client can send “if-modified-since” request, containing the URL of the document and a timestamp. If the document has been modified since the timestamp, the server sends the status code “200” and the modified new data, otherwise the server returns the status code “304”, which stands for document unmodified [141]. Keeping all cached copies coherent with the original server is very challenging due to heterogeneity of the WWW [142]. Figure 2.10 shows the flow diagram of web caching for maintaining consistency of cached objects. Staleness factor [143] is given in equation 2.10.
Caching a document needs the following rules to be followed for keeping the cacheable document fresh in cache [54].

1. When accessing a web document, check the response headers. If the response headers tell the cache not to keep it, then that web document won’t be kept in cache.

2. If no validator such as ETag or Last-Modified header is present on a response, it will be considered uncacheable.

3. The response of authenticated or secure requests won’t be cached.

4. A cached object is considered fresh only if
   
   (a) It has an expiry time or age controlling header set and is still within the fresh period.
(b) If a browser cache has already accessed the object and has been set to check once a session.
(c) If a proxy cache has accessed the object recently and it was modified relatively long ago.

5. Stale objects are refreshed by validating with the origin server and keeping the fresh copy in the cache.

2.9.1 Classes of Responses

The different types of responses in maintaining cache consistency \([46, 144]\) are given below.

- **Fast validations**: Cache returns only a validation of freshness to the requester who presumably already has a copy of the data. (HTTP code 304 Not Modified).
- **Slow validations**: Cache returns a validation of freshness after contacting the origin server and learning that the cached copy is consistent with the original version of the object.
- **Consistency misses or stale hits**: Since cache had stale data, cache returns a new copy of the object after contacting the origin server and getting a fresh copy of the object.
- **Regular misses**: Objects not contained in cache is fetched from server or neighbouring proxy, cached it and returned it to the client.
- **Capacity misses**: Objects were available in cache formerly but had been evicted by the cache replacement policy.
- **Direct**: cache determines the non cacheable and dynamic requests and these are directly send to the origin server. The cache does not keep a copy of these objects.
• **False Misses**: The document requested is cached in any remote cooperating proxies but the summary of the requested proxy does not reflect the fact.

• **False Hits**: The document requested is not cached in any remote cooperating proxies but the summary of requested proxy contains the indication of the availability of that document in remote cache.

### 2.9.2 Types of Consistency Mechanisms

Strong consistency, weak cache consistency, delta consistency and mutual consistency are the different schemes that provide consistency. Self-adaptive Protocol (SATTL), Hybrid Self-Adaptive WWW Cache Coherence Protocol (hSATTL), Web Content Distribution Protocol (WCDP), Adaptive coherence-replacement protocol and Quality of Service (QOS) approaches are hybrid consistency mechanisms or protocols. The mechanisms used for providing the degree of consistency between the client or intermediary and server, falls into three categories: client driven, server driven and explicit mechanisms. Server driven mechanisms referred to as server-based invalidation are used to provide strong or delta consistency guarantees. Server based invalidation requires the server to notify proxies when the data changes. The client driven approach is also referred to as client polling, it polls server periodically to make the objects in cache as fresh. Explicit mechanisms provide specified life time of objects to the client or server explicitly.

#### 2.9.2.1 Weak Consistency

It is a consistency model in which a stale document might be returned to the user. Weak consistency is not always gives satisfaction to the user. Users have to
be aware that the browser may occasionally display a stale page. User has to instruct the browser to “reload” to make sure that a requested document is up-to-date. Reloading of web pages burdens the user as well as web server. Various weak consistency mechanisms are given below.

• **Time-To-Live (TTL):** TTL means the time during which the cached data item is expected to be valid. TTL fields are useful for providing information with known lifetime. A client considers a cached copy up-to-date if time-to-live has not expired. TTLs are simple to implement in HTTP using the optional “expires” header field [145]. In TTL approach, it is often hard to assign an appropriate time-to-live for a document. If TTL value is too large, the probability that the user will see a stale copy of the document increases. If the value is too small, the server will be burdened with many “if-modified-since” messages (IMS), even the document is not changed. Fixed TTL (FTTL) is a server-oriented cache coherence protocol. A cached copy is considered valid until its TTL expires. The server predicts the TTL attribute separately for each object. It is difficult to assign a fixed value for the TTL parameter in advance. Update-risk based TTL estimation method provides the probability that the original data will be updated within the estimated TTL [146].

• **Adaptive TTL (ATTL):** It is also called Alex protocol. It is a proxy-oriented protocol. In adaptive TTL approach, the documents time-to-live is adjusted based on the age of the object. In adaptive TTL, document's time-to-live is calculated from the current time minus the last modified time of the document [144]. Adaptive TTL keeps the frequency of stale documents low and it does not eliminate its occurrence.
• **Client Polling:** A client or intermediary periodically contacts web servers to verify the freshness of cached copies. It polls the server to determine if the data has changed. Frequent polling imposes a large message overhead and also increases the response time. The advantage is that it does not require any state to be maintained at the server and server does not need to delay for modifying web pages. Maintaining consistency is the duty of intermediary or clients. In client polling, it is possible that cache may return stale data and the cache may invalidate data that are still valid [145].

• **Piggyback Invalidation:** In piggyback cache validation (PCV) approach, whenever a proxy has data to communicate to the web server, a proxy cache piggybacks a list of cached objects, potentially stale, from that server for validation. In piggyback server invalidation (PSI) approach, web servers piggyback a list of resources that have changed since it is last accessed by a proxy. A hybrid approach is also available, which combines both PCV and PSI mechanisms used for validating cache contents explicitly [147].

### 2.9.2.2 Strong Consistency

It is a consistency model in which no stale copy of the modified document will ever be returned to the user. Due to the unbounded message delays in the Internet, no consistency mechanism can be strongly consistent. Strong consistency is usually implemented using a two-phase message exchange along with timeouts to handle unbounded delays [148]. Various strong consistency mechanisms are given below.

• **Invalidation approach:** The web server keeps track of all the client sites that have cached copy of document. When the document is changed, it sends the
invalidation messages. This approach relies on server to send notifications when a file is modified. It is difficult for a server to maintain the list of client that has accessed and cached the objects from that server and send invalidation messages to all these clients.

- **Polling-every-time approach**: The cached copy is validated every time when the user requests a document. In this approach, the client sends “if-modified-since” message to server for each cache hit. The disadvantage of this approach is that the user has to wait for a network round trip latency on document retrieval.

- **Adaptive Leases**: Leases provides a trade-offs between the state space overhead and the number of control messages exchanged. The server grants lease to each request from a proxy. The lease duration denotes the interval of time during which the server agrees to notify the proxy if the object is modified. Server and network overhead determines the duration of the lease. After the expiration of the lease, the proxy must send a message to the server for requesting to renew the lease. Leases can be granted to a collection of objects as volume leases [149].

- **MONARCH (Management of Objects in a Network using Assembly, Relationships and Change Characteristics)**: MONARCH identifies relationships among objects composing a page and uses these relationships to keep all objects strongly consistent. It does not require servers to maintain per-client state [150].

### 2.9.2.3 Delta Consistency

Delta consistency is a consistency level that returns data that is never outdated by more than $\delta$ time units, where $\delta$ is configurable parameter. The value of delta should be larger than network delay ($t$) between the server and the intermediary at that instant [148], $t < \delta \leq \infty$. An advantage of delta consistency is that it
provides a quantitatively characterized guarantee by providing an upper bound on the amount by which a cached object could be stale. Another advantage is the flexibility of choosing a different value of δ for each object, allowing the guarantee to be tailored on a per-object basis.

2.9.2.4 Mutual Consistency

Mutual consistency provides consistency guarantee in which a group of objects are mutually consistent with respect to each other [148]. Maintaining consistency of individual objects are not sufficient, a proxy should employ additional mechanisms to ensure that related web objects are mutually consistent with one another. Mutual consistency is useful when a set of objects at a client needs to be consistent with each other. They are automatically invalidated either reflecting the new state or remaining in the earlier stale state.

2.9.2.5 Self-adaptive Protocol (SATTL)

SATTL is used for maintaining coherence of replicated objects in wide area networks. The invalidation protocol and TTL based protocol is combined to improve coherence and to reduce network traffic. In SATTL protocol, TTL is adjusted dynamically for each object [151]. The adjustment is based on the validation check result of IMS messages and on received invalidation requests. After the copy of object is cached, the proxy independently either decreases or increases the value of TTL according to the changes of the original object. If the TTL has not expired and the proxy receives an invalidation message from server, then proxy will decrease the TTL. If the proxy sends an IMS message to the server
and the server responds that the object has changed, TTL is decreased. TTL is increased in following three ways.

1. The proxy sends an IMS message to the server and the server responds that the object has not changed
2. The proxy sends an IMS message to the server and the server responds that the object has changed after the TTL expires and the proxy did not receive an invalidation message.
3. The proxy receives the invalidation message after the TTL expires.

Good features of the proxy oriented and server oriented protocols are combined in developing Hybrid Self-Adaptive WWW Cache Coherence Protocol (hSATTL). This protocol dynamically switches between three cache coherence mechanisms: server-oriented write-invalidate, server-oriented write-update and proxy oriented time-to-live [152]. In the server-oriented protocols, cache coherence is achieved by sending invalidate or update message from server to the proxies. In the proxy-oriented protocols, IMS message is send to the server for validating the cached copy. hSATTL protocol is implemented as the extension to the HTTP protocol. In hSATTL, invalidations of cached copies can be made by adding a “INVALIDATE” header to the HTTP. An “UPDATE” header is added to HTTP for servers to recognize the ability of proxy servers to receive updating information. When the server receives requests with “UPDATE” header, it determines whether the update can be sent to the proxy. If the server sends updating information to the proxy, “Content-Type: update” header is included in the response.
2.9.2.6 Web Content Distribution Protocol (WCDP)

WCDP is an invalidation and update protocol to provide cache consistency for a large number of frequently changing web objects. WCDP supports different levels of consistency such as weak, strong and mutual [153]. WCDP enhances scalability by grouping objects and messages together and by using hierarchical organization for message delivery. In WCDP, objects are grouped into object-groups and multiple messages are aggregated into single message-group. The protocol can invalidate an individual object or set of objects addressed as a unit in an object group. A drawback of this approach is that sending updates incurs a large network overhead especially for large objects. WCDP supports invalidation of immediate-refresh and delayed-refresh by using refresh directives.

2.9.2.7 Adaptive Coherence-Replacement Protocol

Coherence-replacement scheme assigns a replacement priority value to each cache block for deciding which block to remove. This approach combines coherence protocols (write-update and write-invalid protocols) and replacement policies (LRU, LFU etc.) based on network traffic, application execution time, data consistency etc. to optimize the overall performance [154]. This scheme uses an adaptive replacement strategy based on the information such as reference history, access frequency and objects/blocks size. This scheme does not increase the space/time requirements. The goal of this protocol is to provide effective utilization of the distributed cache memory and good application performance.
2.9.2.8 Quality-of-Service (QOS) Approach

A quality-of-service approach is used for providing strong consistency for those data items that require it and permitting weaker consistency for less critical data [155]. QOS specification is defined for data objects as follows.

1. QOS = 0 implies no consistency is required. Always use a cached copy, if it exists.
2. 0 < QOS < 1 implies some level of consistency is required. If the probability of changing of cached object is less than the QOS constraint, use the cached copy.
3. QOS = 1 implies strict consistency. Always validate a cached copy.

2.10 Practical Issues in Cache Design

HTTP provides a standard interface for applications to utilize caches. But HTTP support for caching was limited. It does not provide adequate support for explicitly managing the contents of a cache. Web caches can be implemented at the application level, kernel level or under an embedded operating system [148]. Designing of application level caches are easy and have the potential for most features. Application programs can be used to explicitly add, delete and update cached objects. Kernel level caches are harder to design but have the potential for better performance. Caches designed under embedded operating systems may be optimized for certain features such as communication. It may not be feasible for the embedded operating system to keep up with new processors and their performance may decrease [148]. Web caches need more sophisticated consistency and replacement policies. A number of replacement algorithms and
consistency mechanisms have been proposed. But all techniques have own advantages and disadvantages.

Caches can be implemented in both main memory and disk storage. Main memory offers better performance than disk storage due to the delay of disk access. But disk storage is essential to store cached objects in a non volatile medium for persistence when a system must be shut down and restated later. The performance is likely to be poor, if the cached objects are purged each time the machine is shut down. If the cached objects are stored on disk, the cache can be brought to a warm state easily after the system is restarted. If the cache size exceeds the main memory size, the colder objects are stored in disk instead of deleting the objects to keep the cache within memory limits. When a cache fails, the cache can be quickly brought to a warm state after the failure if hot objects are maintained on disk. File system and databases can be used for persistently storing cached data. But they are inefficient and the rate at which objects are added and deleted can be high. If a file system is used and different files are used for storing each object, the overhead for creating and deleting files can be significant. If a database is used, it is difficult to store objects of different formats in the database. Good performance can be achieved by maintaining multiple objects in a single file and efficiently managing the storage space within the file.

2.11 Squeezing more Performance out of Caches

Researchers and vendors have developed several techniques to extend the utility of web caches. Basic principle of these extended mechanisms is to exploit

- **Clues about Future References**

  Cache uses the reference stream information for making predictions about future references. This information may allow cache to make better replacement decisions. This may also allows cache to more accurately pre-fetch data before it is actually referenced.

- **Filling in or Replacing Gaps**

  An HTTP transfer can terminate in midstream due to a network error or the user clicks the stop button or user clicks another links before the entire page is loaded. Cache can fill the missing data using a range retrieval request using HTTP protocol and subsequent request for the same URL can be satisfied from the cache.

- **Delta Encoding**

  Many frequently referenced web resources changes too frequently, but the changes are often minor. If cache miss occurs due to stale hits, instead of fetching the entire document from the origin servers, it would be efficient to transfer the differences (or delta) between the cache entry and the current resource instance. The use of delta encoding in HTTP requires relatively little protocol overhead and requires a modest amount of computation to create and decode the deltas.

- **Alias Discovery and Duplicate Suppression**

  In the web, same content may appears under more than one URL. There are multiple aliases for a given piece of data. Retrieving and storing multiple copies
of such aliasing could be avoided if cache has the capability of detecting aliases. Additional computation or storage costs may be needed for keeping the alias list and a few HTTP headers or messages may be transferred at the expense of bandwidth for identifying the aliases. Message digest algorithms can be used for finding duplicates by comparing the digest values because it is difficult to generate identical digest values for two different inputs. Such hint information can be send from clients to servers in order to avoid fetching of duplicate information from servers.

2.12 Limitations of Web Caching

Some responses cannot be cached due to their dynamic nature. Fragmentation of dynamic objects into static and dynamic fragments causes caching of static fragments. It is difficult to separate static fragments automatically. Maximizing the cache hit ratio alone does not guarantee the best client response time in the web environment. The objects that are stored in one cache need not be stored in other caches and these objects can be shared among the cooperative caches for the disk space utilization, but it increases the local network traffic by fetching the objects from other caches. One drawback of caching is the potential of using an out-of-date object stored in a cache instead of fetching the current object from the origin server [156]. Various coherency mechanisms have been developed for distributed systems. But HTTP has no way to guarantee coherency for a resource except by disabling caching of that resource. Many cacheable resources are change rapidly. HTTP allows the origin servers to limit the life time of these cacheable resources. But if that objects changed rapidly before its time to live expired, it causes incoherency.
The web is so large that many pages will never be referenced more than once in
the reference stream. Studies have shown that page reference frequencies follow a
distribution similar to Zipf's law, in which the relative probability for a reference
to the Kth most popular page is proportional to 1/K^a, for some constant a, taking
some value less than unity[157]. This implies that most of the web pages are
unlikely to be referenced again in the same reference stream, if large amount of
web pages are available in the universe [4]. Cache has finite storage capacity. If
cache is full, less frequently accessed old entries are removed to make room for
placing new entries. The cache replacement is based on some policy and several
algorithms are available for cache replacement. Difficulties in selecting a
replacement policy [158] are given below.

- Cache contains documents of different sizes
- Request stream consists of request of several users
- Replacement policy imposes implementation costs such as meta-data
  storage, update and lookup costs

Since working set of a large cache does not fit into an economically reasonable
amount of RAM, most of the cached data are stored on disk. The disk storage
adds access latency and queuing delays for retrieving an object from disk. Storage
capacity of RAM and disk are growing exponentially. These growth rates could
increase the effectiveness of caches. But web reference rates also grow faster. Due
to volatile nature of RAM, it is not feasible to build RAM only cache. Combination
of RAM and disk storage is necessary for storing cacheable objects [4]. There is no
guarantee that the cache site has a faster file system, higher bandwidth or a better
network route to the client [159]. Feldmann et al. shown that almost 30% of requests had cookies, which dramatically limits the number of requests that can be satisfied from a proxy cache [37]. Studies shown that 35% to 50% of web documents are uncacheable because their content is specific to the initial request and many documents are requested only once [160]. If the web server could transmit information about reference inter arrival time, modification intervals, correlation between references etc. to the caches, these hints would be useful for caches to optimize their behavior. The current HTTP protocol provides no mechanism for providing such hint information [161].

2.13 Concluding Remarks

This chapter has been an attempt to give an overview of the important topics in web caching, which constitute the background knowledge of the research conducted. Web caching allows the retrieved documents to be kept close to clients for improving the performance of the web. Many studies shown that cache hit ratio achieved by web caching is up to 50% and latency reduction is up to 28% [33, 34, 37]. There is scope for further improvement in web caching for increasing cache hit ratio and reducing latency further.