Design Considerations for Integrated Proxy Servers

Sambit Sahu, **Prashant Shenoy** and **Don Towlsey**

Department of Computer Science, University of Massachusetts, Amherst, MA 01003.
Email: {sahu,shenoy,towsley}@cs.umass.edu

1 Introduction

The growth of the Internet and the World Wide Web has enabled an increasing number of users to access vast amounts of information stored at geographically distributed sites. Due to the non-uniformity of information access, however, popular objects create "hot-spots" of server and network load, and thereby significantly increase latency for information access [18]. Proxy servers provide a way to partly alleviate these overheads. In such an architecture, clients request objects from a proxy; the proxy services client requests using locally cached data or by fetching the requested object from the server. By caching frequently accessed objects and servicing requests for these objects from the cache, proxies can reduce the load on network links and servers, as well as reduce client access latencies.

Recently several proxy servers have been designed to service web requests consisting of textual and image objects [3, 10, 17]. However, the web is rapidly evolving from a predominantly text (and image) based information system to a full-fledged multimedia information system. A recent study has shown that the number of audio and video (*continuous media*) objects stored on web servers tripled in the first nine months of 1998 [8]. Although continuous media objects constitute a small fraction of the data currently stored on such servers, it is estimated that, by 2003, more that 50% of the data stored on servers will be continuous media [4]. Furthermore, clients accessing this data are expected to range from hand-held PDAs to high-end workstations, each having different capabilities and service requirements. Current web proxies designed for textual and image data neither cache continuous media data, nor deal with the diversity in service requirements of users. Consequently, proxy servers need to be extended along several dimensions to efficiently support the myriad of present and future applications. The key challenge in designing such proxy servers is that they need to deal with heterogeneity in data characteristics as well as heterogeneity in the service requirements of applications. We refer to such a proxy as an *integrated proxy server*. In this paper, we examine the architecture and mechanisms required for designing integrated proxy servers. Specifically, we argue that an integrated proxy should: (i) employ a diverse set of mechanisms to efficiently support heterogeneous clients, (ii) allow these mechanisms to be dynamically composed to provide a customized per-client service, (iii) efficiently handle a multi-resource cache consisting of memory and disk, possibly by monitoring the workload characteristics, and (iv) employ flexible scheduling and resource management policies to maximize throughput and utility to users.

The rest of this paper is organized as follows. Section 2 reviews existing proxy servers and identifies their limitations. Section 3 evaluates architectural alternatives for designing integrated proxy servers. In Section 4, we examine the requirements imposed by integrated proxy servers. Section 5 briefly describes a proxy server architecture that meets these requirements.

2 Inadequacies of Existing Proxy Servers

Recently several proxy servers that handle conventional web (text and image) requests have been designed [3, 10, 17]. Since such proxies cache large amounts of data, they employ disk-based caches and use conventional file systems to store and retrieve data from disk. Since conventional web requests desire low average response times but no absolute performance guarantees, the best-effort service provided by such file systems suffices for web requests. Moreover, these servers exploit locality in web accesses by employing cache replacement policies such as LRU to maximize the hit ratio.

Continuous media have significantly different characteristics as compared to conventional data (with respect to size, data rate, timeliness, etc), and hence, many of the mechanisms employed by existing proxy servers are unsuitable for such requests. To illustrate, continuous media requests impose real-time constraints on the storage and retrieval of data to ensure jitter-free playback. Consequently, employing a conventional (best-effort) file system to manage the disk cache is inadequate for this purpose. Moreover, the server-push (streaming) paradigm is more suited to continuous media requests, which is fundamentally different from the client-pull paradigm employed by existing proxies to service conventional requests. Finally, continuous media accesses are predominantly sequential in nature. Cache replacement policies such as LRU employed by existing proxy servers are known to be ineffective for sequential accesses [2].

Several research groups have investigated the design of specialized continuous media proxies to alleviate some of the drawbacks of conventional web proxies. Most of these efforts have focussed on designing mechanisms for handling continuous media requests and several mechanisms such as prefix caching, forward error correction, smoothing, batching and transcoding have been proposed [1, 7, 12, 13]. Most continuous media proxies support user requests by employing a subset of these mechanisms and appropriately parameterizing them to specific needs. However, the increasing heterogeneity in the service requirements of users has made it difficult, if not impossible, to support a diverse workload using a small set of mechanisms. Consequently, a proxy will need to employ a rich set of mechanisms to support heterogeneous clients. Furthermore, it will need to allow its service to be tailored to the user needs (by allowing various mechanisms to be dynamically combined to create a customized per-client service). Existing continuous media proxies neither allow dynamic composition of mechanisms to create a customized service, nor do they allow modular extensions to the set of supported mechanisms. Second, much of the effort in designing continuous media proxies has focused on designing mechanisms for efficiently handling user requests, and the issue of managing continuous data on a disk-based cache has not received much attention. Finally, continuous media proxies are designed for audio and video requests and typically do not handle conventional web requests.

Some recently released commercial proxy servers employ features that allow them to manage both continuous media and conventional web requests [9, 11]. Such servers also employ specialized placement techniques, rather than general purpose file systems, to store and retrieve objects from disk, which allows them to improve server throughput. However, even these state of the art proxy servers employ a fixed set of mechanisms to handle various classes of requests and do not allow composition of various mechanisms to provide customized service.

In summary, existing proxy servers are unsuitable for managing a heterogenous clientele accessing data with diverse characteristics. This motivates the need for designing an integrated proxy server to address these limitations. Next, we investigate architectural alternatives of such a proxy server.

3 Design Methodologies

There are two methodologies for designing integrated proxies: (i) a *logically integrated* architecture, that employs separate proxies for each application class and uses an integrated layer to provide a logically unified view to applications; and (ii) a *physically integrated* architecture that employs a single proxy to service all application classes. Since techniques for building proxies optimized for a single application class are well known (e.g., conventional web proxies, video proxies), logically integrated proxy servers are simple to design and implement. However, static partitioning of resources among component proxy servers inherent in this approach can lead to under-utilization of resources, especially in scenarios with dynamically fluctuating workloads. Moreover, since service requirements can vary even within an application class (e.g., loss-tolerant video, delay-intolerant video, etc) implementing a separate component proxy for each sub-class can further exacerbate this problem. Since physically integrated proxies dynamically share their resources across various classes, they yield better utilization and potentially better application performance. However, dynamic multiplexing of resources requires the proxy to employ complex resource management and scheduling techniques.

Several studies have investigated these two methodologies in the context of integrated file systems and integrated services networks [14, 15]. These studies have shown, both qualitatively and quantitatively, the benefits of dynamic resource allocation over static partitioning, and have concluded that the benefits of improved resource utilization due to physical integration more than offset the drawback of increased complexity. We conjecture that a similar result holds for integrated proxy servers and are in the process of conducting a performance evaluation of the two methodologies to verify our hypothesis. Due to its inherent advantages, we choose the physically integrated architecture for designing our integrated proxy server. In fact, newest versions of several commercial proxy servers, such as Inktomi's Traffic Server [9], also employ the physically integrated architecture. However, as explained in Section 2, these proxy servers lack several features desirable for managing heterogeneity in data and application requirements.

4 Requirements for Integrated Proxy Servers

A physically integrated proxy server should achieve efficient utilization of server resources while managing heterogeneity in application requirements and data characteristics. Meeting these objectives imposes several requirements on the proxy server.

- *Composability:* To efficiently support clients with different service requirements, an integrated proxy server should tailor the service provided to meet needs of individual clients. To achieve this objective, the proxy server should employ a rich set of mechanisms and *allow these mechanisms to be dynamically composed to create a customized per-client service.* Thus, smoothing and transcoding could be combined to service a low-bandwidth client, whereas forward error correction and prefix caching mechanisms could be combined to service a client that has low delay and loss tolerance. The proxy server architecture should also facilitate easy addition of new mechanisms to support requirements of future clients as well as allow newly added mechanisms to be composed with the set of existing mechanisms.

- *Multi-resource cache management:* Since an integrated proxy caches large amounts of data, it employs a disk-based cache to store these objects and employ a smaller memory cache to improve latencies for frequently accessed objects. Managing such a multi-resource cache necessitates the development of novel cache replacement and cache management policies that take advantage of the workload characteristics. Whereas the cache replacement policy determines *which* objects to store in the cache (and which ones to evict), the cache management policy determines *where* to store an object in the cache (e.g., memory, disk, both). A cache replacement policy suitable for integrated proxy environments should take into account diverse sizes of objects as well as differences in access characteristics of objects when making replacement decisions. Most existing cache replacement policies deal with homogeneous objects and access characteristics and hence, are unsuitable for this purpose. The design of a cache management policy suitable for integrated proxies has not received much attention in the literature. The key challenge in designing such a policy is to develop *efficient techniques to monitor the workload and maintain access statistics so as to aid its decisions.* Based on these statistics, the policy may prefetch objects from disk to memory in anticipation of their access. The policy may also have to trade one resource against another when making decisions on storing and migrating objects. For example, it may decide to store large hot continuous media objects on disk, rather than in memory, and utilize the cache space in memory to store a large number of small text objects. Finally, the policy may decide to store portions of an object (e.g., continuous media prefixes) in memory and the remainder on disk to reduce access latencies.

- *Scheduling and resource management:* Since an integrated proxy server services requests with different requirements, it must ensure that these requests do not interfere with each other. For instance, real-time continuous media requests must not affect the response times of best-effort web requests, and a burst of best-effort web requests must not affect the real-time guarantees provided to continuous media requests. Hence, the scheduling algorithm employed by the proxy should protect various classes from one another, while providing all the benefits of dynamic resource sharing. Furthermore, each client request arriving at the proxy triggers one or more cache requests (especially long-lived continuous media requests that periodically retrieve data from the cache for steaming). In such a scenario, the scheduling algorithm should ensure that it aligns the service provided with the requirements of individual cache requests.

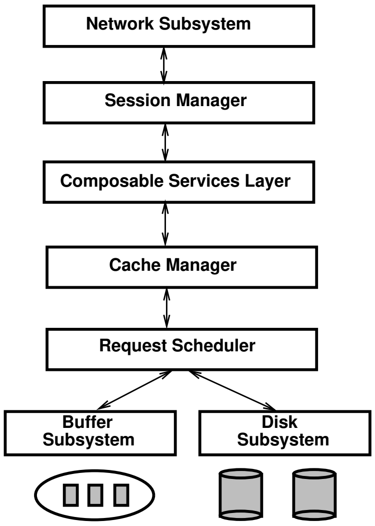

Figure 1: Architecture of our integrated proxy server

In general, resource management techniques employed by the proxy must address the challenge of integration—efficiently managing requests and data with diverse characteristics. To do so, the proxy can employ a single integrated technique for each resource to manage all classes. Alternatively, the proxy can allow multiple data-type specific techniques to manage a resource and employ mechanisms that enable their coexistence.

- *Operating system issues:* An integrated proxy can provide performance guarantees to applications only in conjunction with an operating system that can itself allocate resources in a predictable manner. Thus, we assume that the underlying operating system employs predictable resource allocation techniques for resources such as CPU, disks and network interfaces. Second, unlike video and web servers that are dominated by read requests, a significant portion of the workload at a proxy consists of writes and deletes (resulting from evictions of cold objects and fetching new objects into the cache). Consequently, the operating system must employ placement policies that minimize fragmentation of disk space resulting from frequent writes and deletes. Moreover, the placement policy should efficiently handle the storage of diverse objects ranging from small text files to large continuous media files. Most existing placement policies have been either developed for predominantly read-only continuous media workloads in video servers or for small textual files stored in conventional file systems; hence, these polices are unsuitable for proxy workloads.

5 Architecture of an Integrated Proxy Server

We are designing an integrated proxy server that meets the requirements outlined in Section 4. Figure 1 depicts the key components of our architecture. A novel feature of our architecture is the *composable services layer* that provides facilities to dynamically compose mechanisms as well as to add support for new mechanisms. Conceptually, each mechanism in this layer either transforms (modifies) the request stream or the data stream; a sequence of mechanisms can then be combined to provide customized service. For instance, smoothing mechanisms modify the request schedule to smooth out bit rate variations, whereas transcoding mechanisms modify the data stream; together they can provide smoothed transcoded continuous media streams to users. Besides the composable services layer, our architecture consists of a number of other components such as: (i) the *network subsystem* that provides interfaces (e.g., http) to facilitate client-proxy and proxy-server communication, (ii) a *session manager* that manages and maintains state of active client sessions, (iii) a *cache manager* that instantiates cache management and cache replacement policies to manage a two level cache consisting of memory and disk, (iv) a *request scheduler* that partitions the server bandwidth fairly across classes while meeting requirements (e.g., deadlines) of individual requests and (v) *buffer and disk subsystems* that manage storage and retrieval of objects from memory and disk, respectively.

Separately, we are also designing an QoS-enhanced Linux kernel (jointly with AT&T Research and the Univ. of Texas) that can allocate resources in a predictable manner. Specifically, the kernel employs: (i) the H-SFQ CPU scheduler that allocates CPU bandwidth fairly among application classes [5], (ii) the SFQ link scheduler that can fairly allocate network link bandwidth to network flows [6], and (iii) the Cello disk scheduler that can support disk requests with diverse requirements [16]. We plan to use this QoS-enhanced Linux kernel as the substrate for our integrated proxy server.

In conclusion, the growing use of continuous media objects in the web have made conventional proxy servers inadequate. In this paper, we focused on the design of integrated proxy servers that addresses this limitation. We investigated two methodologies for designing such servers and then listed several requirements that must be met by integrated proxies. Finally, we briefly described the architecture of an integrated proxy server that is being built in our research group.

References

[1] E. Amir, S. McCanne, and R. Katz. An Active Service Framework and Its Application to Real-time Multimedia Transcoding. In *Proceedings of ACM SIGCOMM Conference, Vancouver, Canada*, pages 178–189, September 1998.

[2] P. Cao. *Application Controlled File Caching and Prefetching*. PhD thesis, Princeton University, 1996.

[3] A. Chankhunthod, P B. Danzig, C. Neerdaels, M F. Schwartz, and K J. Worrell. A Hierarchical Internet Object Cache. In *Proceedings of the 1996 USENIX Technical Conference, San Diego, CA*, January 1996.

[4] G. A. Gibson, J.S. Vitter, and J. Wilkes. Storage and I/O Issues in Large-Scale Computing. *ACM Workshop on Strategic Directions in Computing Research, ACM Computing Surveys*, 1996. http://www.medg.lcs.mit.edu/doyle/sdcr.

[5] P. Goyal, X. Guo, and H. Vin. A Hierarchical CPU Scheduler for Multimedia Operating Systems. In *Proceedings of the Second Symposium on Operating Systems Design and Implementation*, pages 107–121, October 1996.

[6] P. Goyal, H. M. Vin, and H. Cheng. Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks. In *Proceedings of ACM SIGCOMM'96*, pages 157–168, August 1996.

[7] K A. Hua, Y. Cai, and S. Sheu. Patching: A Multicast Technique for True Video-on-Demand Services. In *Proceedings of 6th ACM Conference on Multimedia, Bristol, UK*, pages 191–200, September 1998.

[8] Streaming Media Caching White Paper. Technical report, Inktomi Corporation, Available on-line at http://www.inktomi.com/products/traffic/tech/streaming.html, 1999.

[9] Traffic Server Product Details. Inktomi Corporation, http://www.inktomi.com/products/traffic, 1999.

[10] C. Maltzahn, K. Richardson, and D. Grunwald. Performance Issues of Enterprise Level Web Proxies. In *Proceedings of the SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, June 1997.

[11] NetCache Product Details. Network Appliance, Inc., http://www.netapp.com/products/internet_prod.html, 1999.

[12] J. Salehi, Z. Zhang, J. Kurose, and D. Towsley. Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements through Optimal Smoothing. In *Proceedings of ACM SIGMETRICS, Philadelphia, PA*, May 1996.

[13] S. Sen, J. Rexford, and D. Towsley. Proxy Prefix Caching for Multimedia Streams. In *Proceedings of the IEEE Infocom'99, New York, NY*, March 1999.

[14] S. Shenker. Fundamental Design Issues for the Future Internet. *IEEE Journal of Selected Areas in Communications*, 13:1176–1188, September 1995.

[15] P. Shenoy, P. Goyal, and H M. Vin. Architectural Considerations for Next Generation File Systems. Technical Report TR98-48, Dept. of Computer Science, Univ. of Massachusetts at Amherst, 1998.

[16] P Shenoy and H M. Vin. Cello: A Disk Scheduling Framework for Next Generation Operating Systems. In *Proceedings of ACM SIGMETRICS Conference, Madison, WI*, pages 44–55, June 1998.

[17] *Squid Internet Object Cache Users Guide*. Available on-line at http://squid.nlanr.net, 1997.

[18] R. Tewari, M. Dahlin, H M. Vin, and J. Kay. Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet. In *Proceedings of the 19th International Conference on Distributed Computing Systems (ICDCS) (to appear)*, June 1999.