APPOINT: An Approach for Peer-to-Peer Offloading the INTernet *

Egemen Tanin
Hanan Samet
Computer Science Department
Center for Automation Research
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742
(301) 405-0368
{egemen,hjs}@umiacs.umd.edu
www.cs.umd.edu/{~egemen,~hjs}

Abstract

The Internet has recently become the medium of interaction with large volumes of online data. Enterprises in the public and private sectors made their data archives available over the Internet. Working with such large volumes of online data is a challenging task. APPOINT (an Approach for Peer-to-Peer Offloading the INTernet), a centralized peer-to-peer approach that helps users of the Internet transfer large volumes of online data efficiently, is introduced with this paper. In APPOINT, active clients of a client-server architecture act on the server's behalf and communicate with each other to decrease network latency, improve service bandwidth, and resolve server congestions. In essence, a server is enabled to appoint alternatives when needed and create a scalable collaborative virtual mirror from the active clients. Multiple parameters such as locality and availability to decide on how to best forward a download request are considered. The problems that can arise in symmetric upload requests are also addressed. APPOINT is an ongoing project and it is being built as an add-on to existing client-server systems. A library of functions, with a simple application programming interface (API) that can be used within an existing client-server system to improve the service, is currently being developed.

1. Introduction

In recent years, many enterprises in the public and private sectors have provided access to large volumes of data over the Internet. Downloading, processing, and finally uploading large volumes of data is a challenge. We have been developing a sample interactive data browser for access to online databases. Specifically, our application domain contains large volumes of online spatial data that require high levels of visual interactions. We define two different types of usages for our browser. First, the browser can be activated as an applet so that users across various platforms can continuously access large spatial data on a remote location without having to install any additional

*This work was supported in part by the National Science Foundation under Grants EIA-99-01636, EIA-99-00268, IIS-00-86162, and EIA-00-91474.

software on their side. Second, the browser along with an Internet-enabled database management system can be installed on the local architecture. In this case, the browser can still be utilized to view data from remote locations. However, in this second type of usage, frequently used data can be downloaded to the local database on demand, and subsequently accessed locally. Users can process and then upload large volumes of spatial data back to the server using this new browser.

We focus our research efforts in two directions. First, we are developing efficient caching methods to balance local resources on one side and the significant latency of the network connection on the other. The low bandwidth of this connection is the primary concern in both cases. The outcome of this research will primarily address the issues of our first type of usage (as an applet), for our browser and other similar applications. Second, we want to help users that wish to manipulate large volumes of online data for prolonged periods of time by developing APPOINT (an Approach for Peer-to-Peer Offloading the INTernet), a centralized peer-to-peer approach to provide them with the ability to transfer large volumes of data more efficiently by better utilizing the distributed network resources among active clients of a client-server architecture. The results of this research addresses primarily the issues of the second type of usage for our browser. Although we are using a spatial data browser to test our approach, we believe that our approach is highly generalizable to other client-server applications. This paper focuses on APPOINT and on large online data transfer aspects of our research using peer-to-peer systems.

The rest of this paper is organized as follows. Section 2 describes our approach, APPOINT, in more detail. It also introduces our developing application programming interface (API). Section 3 discusses APPOINT in relation to existing work. Section 4 gives a sample client-server scenario with APPOINT. Section 5 contains concluding remarks as well as a discussion of our plans for the future.

2. Our Developing Approach: APPOINT

Optimizing the network download and upload times for large volumes of data is our main goal. The common client-server approach to transferring data between the two ends of a connection assumes a designated role for each one of the ends, i.e., some clients and a server. It also ignores the fact that the needs of the two ends may be time dependent, i.e., there can be congested periods of usage for the server while multiple clients may be idle and waiting to be served. There is also a common understanding that data moves over a single pipe, i.e., from a server to a client. With the recent changes in the Internet Community, one can imagine a peer-to-peer approach instead, where the two ends (peers) can assume both the roles of a client and a server from time to time, promising to improve the overall network performance, the most scarce of resources needed for transferring large volumes of online data.

We are building APPOINT as a centralized peer-to-peer system to demonstrate our approach for improving client-server systems. A company or a government server still exists. There is a central source for data and a decision mechanism for the service. The environment still functions as a client-server environment under many circumstances. Yet, unlike many other client-server environments, APPOINT maintains more information about the clients. This includes, inventories of what each client downloads, their networking capabilities, locations, and availabilities. When the client-server service starts to perform poorly or a request for a data item comes from a client with a poor connection to the server, APPOINT can start appointing appropriate active clients of the system to serve on behalf of the server, i.e., clients who have already volunteered their services and can take on the role of peers (hence, moving from a client-server scheme to a peer-to-peer scheme). The directory service for the active clients is still performed by the server but the server no longer

serves all the requests. In this scheme, clients are used mainly for the purpose of sharing their networking resources rather than introducing new contents and hence they help offload the server and scale up the service. The existence of a server is simpler in terms of management of dynamic peers in comparison to pure peer-to-peer approaches where a flood of messages to discover who is still active in the system should be used by each peer that needs to make a decision. The server is also the main source of data and under regular availability and connectivity circumstances it may not forward the service.

Data is assumed to be formed of files. A single file forms the atomic means of communication. APPOINT optimizes requests with respect to these atomic requests. Frequently accessed data sets are replicated as a byproduct of having been requested by a large number of users. This opens up the potential for bypassing the server in future downloads for the data by other users as there are now many new points of access to it. Bypassing the server is useful when the server's bandwidth is limited. Existence of a central server assures that unpopular data is also available at all times. The service depends on the availability of the central server. The central server is now more resilient to congestions as the service is more scalable.

Backups and other maintenance activities are already being performed on the central server and hence no extra administrative effort is needed for the dynamic peers. If a peer goes down, no extra precautions are taken. In fact, APPOINT does not require any additional resources from an already existing client-server environment but, instead, expands its capability. The peers simply get on to or get off from a table on the server.

Uploading data is achieved in a similar manner as downloading data. For uploads, the active clients can again be utilized. Users can upload their data to a set of peers other than the main server if the server is busy or lies in a distant location. These decisions are made by a process running on the central server. Of course, eventually when the server is available, the data is propagated to it.

All the operations are performed in a transparent fashion to the clients. Upon initial connection to the server, they are asked whether they want to share their idle networking time and disk space or not. The rest of the operations follow transparently after the initial contact. APPOINT works on the application layer but not on lower layers in order to achieve platform independence and easy deployment of the system. APPOINT is not a replacement but an addition to the current client-server architectures. We are developing a library of function calls that when placed in a client-server architecture starts the service. Location of active clients, bandwidth among active clients, data-size to be transferred, load on active clients, availability of an active client are all used to select the best clients that can form a contemporary alternative to the server.

With APPOINT we are defining a very simple API that could be used within an existing client-server system easily. Instead of denial of service, this API can be utilized to forward the service. The API for the server side is:

```
start(serverPortNo)
makeFileAvailable(fileName, fromLocation, booleanValue)
callback receivedFile(fileName, toDefaultLocation)
callback errorReceivingFile(fileName, toDefaultLocation, errorString)
stop()
```

Similarly the API for the client side is:

```
start(clientPortNo, serverPortNo, serverAddress)
makeFileAvailable(fileName, fromLocation, booleanValue)
receiveFile(fileName, toLocation)
sendFile(fileName, fromLocation)
stop()
```

Many client-server systems define a limit for their services. When this limit is reached, the server denies further service to the incoming clients. In such situations, incoming clients can request the file that they want to download from APPOINT by using the APPOINT API. If there are active clients making this file publicly available, they will get their file from these clients. Similarly, denial of service for large upload requests can be handled by utilizing the APPOINT API. The details of how transfers and assignments are made remain transparent to the designer of the client-server system. To the user, even the existence of APPOINT is transparent during operation. It is possible to use the APPOINT API for many different purposes. A programmer can even write a simple program, i.e., a centralized peer-to-peer file exchange system, by using the APPOINT API.

3. APPOINT's Relation to Existing Work

Peer-to-peer systems are gaining momentum [Oram, 2001] over the Internet. Napster [Napster, 1999] and Gnutella [Gnutella, 2000] were the first and most well-known of these systems. Napster was also a centralized peer-to-peer system like APPOINT where the directory service is centralized on servers and people exchanged music files that they have on their disks. Our application area, where the data is already freely available to the public, forms a prime candidate for a peer-to-peer approach. Gnutella was a pure (decentralized) peer-to-peer file exchange system. Unfortunately, it suffered from scalability issues, i.e., floods of messages between peers to map connectivity in the system are required. Other systems followed these, each addressing a different flavor of sharing over the Internet. Multiple peer-to-peer storage systems have recently emerged. PAST [Rowstron and Druschel, 2001b], Freenet [Clarke et al., 2000], Ficus [Page et al., 1997], Free Haven [Dingledine et al., 2000], Eternity Service [Anderson, 1996], CFS [Dabek et al., 2001], and OceanStore [Kubiatowicz et al., 2000] are some popular peer-to-peer storage systems. Some of these systems, like Freenet, have focused on anonymity while others, like PAST, have focused on persistence of storage. Also, other approaches, like SETI@Home [SETI@Home, 2001], made other resources, such as idle CPUs, work together over the Internet to solve large scale computational problems. Our goal is different than these approaches. We want to improve existing client-server systems in terms of performance by using idle networking resources among active clients. Hence, other issues like anonymity, decentralization, and persistence of storage were less important in our decisions.

Along with these systems, researchers have developed underlying routing and location utilities to address the core problems of mainly pure peer-to-peer systems. These are for optimally locating and accessing an object in a decentralized dynamic distributed system over a wide-area network. Examples of some lower level utility systems are [Plaxton et al., 1999, Rowstron and Druschel, 2001a, Stoica et al., 2001]. Multiple peer-to-peer systems are built on them. For example, PAST is built on the Pastry [Rowstron and Druschel, 2001a] location and routing service. Unfortunately, these systems do not directly fit to our application domain, i.e., one where the data is generated and maintained by a central governmental agency or a company.

From our perspective, although APPOINT employs some of the techniques used in pure

peer-to-peer systems, it is also closely related to current Web caching architectures. Squirrel [Iyer et al., 2002] forms the middle ground. It is also built on top of Pastry and creates a pure peer-to-peer collaborative Web cache among the Web browser caches of the machines in a local-area network. Except for this recent peer-to-peer approach, Web caching is mostly a well-studied field in the realm of server/proxy level caching [Breslau et al., 1999, Cao and Irani, 1997, Challenger et al., 1999, Dingle and Partl, 1996, Karger et al., 1997, Karger et al., 1999, Kurcewicz et al., 1998, Rabinovich et al., 1998, Wolman et al., 1999]. Collaborative Web caching systems, the most relevant of these for our research, focus on creating either a hierarchical, hash-based, central directory-based, or multicast-based caching schemes. We do not compete with these approaches. In fact, APPOINT can work in tandem with collaborative Web caching if they are deployed together. We try to address the situation where a request reaches to a central server, meaning all the caches report a miss. Hence, the point where the server is reached can be used to take a central decision but then the actual service request can be forwarded to a set of active clients, i.e., the download and upload operations. Cache misses are especially common in the type of large data-based services on which we are working on. Most of the Web caching schemes that are in use today employ a replacement policy that gives a priority to replacing the largest sized items over smaller-sized ones. Hence, these policies would lead to the immediate replacement of our relatively large data files even though they may be used frequently. In addition, in our case, the user community that accesses a certain data file may also be very dispersed from a network point of view and thus cannot take advantage of any of the caching schemes. For example, a data file containing a scientific data set may gather interest from thousands of scientists around the world, but the fact that these scientists are dispersed among many different countries can make their local caching schemes ineffective. Finally, none of the Web caching methods address the symmetric issue of large data uploads.

Bistro [Bhattacharjee et al., 2000] is a recent framework for building scalable wide-area upload applications. For example, deadline driven online tax form submissions forms a prime application for this framework. This work is closely related to our approach. It uses intermediaries (in comparison to our active clients), termed Bistros, for improving the efficiency and scalability of uploads. The Bistro approach is to break the original deadline-driven upload problem into the following pieces: (i) a real-time timestamp subproblem, (ii) a low latency commit subproblem, where the data goes to an intermediary, and (iii) a timely data transfer subproblem, which can be carefully planned (and coordinated with other uploads) and results in efficient data delivery to the original destination.

4. A Sample Application

SAND (denoting Spatial And Nonspatial Data) is a prototype spatial database system developed at the University of Maryland [Esperança et al., 2001]. The SAND Internet Browser provides a graphical user interface to the facilities of SAND over the Internet. Users specify queries by choosing the desired selection conditions from a variety of menus and dialog boxes. The SAND Internet Browser forms a good example application to the applications that are used to (interactively) manipulate large volumes of online data.

We envision two different types of usages for the SAND Internet Browser. First, the SAND Internet Browser can be activated as an applet to enable users across various platforms to access a spatial database on a remote computer without having to install the SAND system on their side. Second, the SAND Internet Browser along with the SAND spatial database system can be installed on the client side. In the latter case, the SAND Internet Browser can be utilized to view data

from remote data sources, while frequently used data can be loaded to the local SAND database on demand, and subsequently accessed locally. In time, users can build up their own local databases. They can also transfer local data files (results of their daily work) back to a central server. We will use this local application oriented latter case as our test client-server application for APPOINT.

When users want to download some large data from a remote server, there are three possible cases that can occur: (i) the server is too busy and hence denies service, (ii) the server is available but the connection is poor, or (iii) the server is available and there is a good connection to it. For the first two cases, if the server decides to use APPOINT, the client can be notified of this decision. Then, the client can request the file by using the APPOINT API. When the file is ready, it can be accessed locally by the SAND Internet Browser. Similarly, for an upload action we can create the three cases mentioned above. A client can send the file by again using APPOINT. Yet, in uploads, when the file finally reaches the destination, the server is notified by a call to the relevant callback function. Hence, the client-server system observes two confirmations for the uploads, one when the file goes into the network and one for the receipt of the file by the server. These two notifications can be used separately by the designer of the client-server system. In our case, we explicitly use the first notification to inform the user about the submission, and the second for the receipt by the server. So the transfer can be viewed as a large email sent to an automated system. It is also possible to use the APPOINT API in a more direct manner where the large data file transfers are directly requested from APPOINT even without checking the connectivity or the status of the server. If the server has made all of it's files visible through APPOINT, then APPOINT will choose it when there are not any better alternatives than the server. In many client-server applications this type of APPOINT API usage may be preferred as it is easier to implement.

5. Concluding Remarks

By developing APPOINT, we are introducing a novel approach to improving service under current client-server systems. We envision the development of new efficient algorithms with this work for large online data transfers using active clients of a client-server system. In our approach, a client can go down anytime and hence this requires a special attention and we want to enhance our methods to include dynamic updates.

References

[Anderson, 1996] Anderson, R. (1996). The Eternity Service. In *Proceedings of the PRAGOCRYPT'96*, pages 242–252, Prague, Czech Republic.

[Bhattacharjee et al., 2000] Bhattacharjee, S., Cheng, W. C., Chou, C.-F., Golubchik, L., and Khuller, S. (2000). Bistro: A platform for building scalable wide-area upload applications. *ACM SIGMETRICS Performance Evaluation Review*, 28(2):29–35.

[Breslau et al., 1999] Breslau, L., Cao, P., Fan, L., Phillips, G., and Shenker, S. (1999). Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of the IEEE Infocom'99*, pages 126–134, New York, NY.

[Cao and Irani, 1997] Cao, P. and Irani, S. (1997). Cost-aware WWW proxy caching algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS)*, pages 193–206, Monterey, CA.

[Challenger et al., 1999] Challenger, J., Iyengar, A., and Dantzig, P. (1999). A scalable system for consistently caching dynamic Web data. In *Proceedings of the IEEE Infocom'99*, pages 294–303, New York, NY.

[Clarke et al., 2000] Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. (2000). Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, pages 311–320, Berkeley, CA.

[Dabek et al., 2001] Dabek, F., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. (2001). Wide-area cooperative storage with CFS. In *Proceedings of the ACM SOSP'01*, pages 202–215, Banff, AL.

[Dingle and Partl, 1996] Dingle, A. and Partl, T. (1996). Web cache coherence. *Computer Networks and ISDN Systems*, 28(7-11):907–920.

[Dingledine et al., 2000] Dingledine, R., Freedman, M. J., and Molnar, D. (2000). The Free Haven Project: Distributed anonymous storage service. In *Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, pages 67–95, Berkeley, CA.

[Esperança et al., 2001] Esperança, C., Hjaltason, G. R., Samet, H., Brabec, F., and Tanin, E. (2001). An overview of the SAND spatial database system. University of Maryland, submitted for publication.

[Gnutella, 2000] Gnutella (2000). http://www.gnutella.com/.

[Iyer et al., 2002] Iyer, S., Rowstron, A., and Druschel, P. (2002). Squirrel: A decentralized peer-to-peer Web cache. Rice University/Microsoft Research, submitted for publication.

[Karger et al., 1997] Karger, D., Lehman, E., Leighton, T., Levine, M., Lewin, D., and Panigraphy, R. (1997). Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 654–663, El Paso, TX.

[Karger et al., 1999] Karger, D., Sherman, A., Berkheimer, A., Bogstad, B., Dhanidina, R., Iwamoto, K., Kim, B., Matkins, L., and Yerushalmi, Y. (1999). Web caching with consistent hashing. *Computer Networks*, 31(11-16):1203–1213.

[Kubiatowicz et al., 2000] Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., and Zhao, B. (2000). OceanStore: An architecture for global-scale persistent store. In *Proceedings of the ACM ASPLOS'00*, pages 190–201, Cambridge, MA.

[Kurcewicz et al., 1998] Kurcewicz, M., Sylwestrzak, W., and Wierzbicki, A. (1998). A distributed WWW cache. *Computer Networks and ISDN Systems*, 30(22):2261–2267.

[Napster, 1999] Napster (1999). http://www.napster.com/.

[Oram, 2001] Oram, A. (2001). *Peer-to-peer: Harnessing the power of disruptive technologies*. O'Reilly and Associates, Sebastopol, CA.

[Page et al., 1997] Page, T. W., Guy, R. G., Heidemann, J. S., Ratner, D. H., Reiher, P. L., Goel, A., Kuenning, G. H., and Popek, G. (1997). Perspectives on optimistically replicated peer-to-peer filing. *Software – Practice, and Experience*, 11(1):155–180.

[Plaxton et al., 1999] Plaxton, C. G., Rajaraman, R., and Richa, A. W. (1999). Accessing nearby copies of replicated objects in a distributed environment. *Theory of Computing Systems*, 32(3):241–280.

[Rabinovich et al., 1998] Rabinovich, M., Chase, J., and Gadde, S. (1998). Not all hits are created equal: Cooperative proxy caching over a wide-area network. *Computer Networks and ISDN Systems*, 30(22-23):2253–2259.

[Rowstron and Druschel, 2001a] Rowstron, A. and Druschel, P. (2001a). Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the ACM Middleware'01*, pages 329–350, Heidelberg, Germany.

[Rowstron and Druschel, 2001b] Rowstron, A. and Druschel, P. (2001b). Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the ACM SOSP'01*, pages 160–173, Banff, AL.

[SETI@Home, 2001] SETI@Home (2001). http://setiathome.ssl.berkeley.edu/.

[Stoica et al., 2001] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of the ACM SIGCOMM'01*, pages 149–160, San Diego, CA.

[Wolman et al., 1999] Wolman, A., Voelker, G. M., Sharma, N., Cardwell, N., Karlin, A., and Levy, H. M. (1999). On the scale and performance of cooperative Web proxy caching. In *Proceedings of the ACM SOSP'99*, pages 16–31, Kiawah Island, SC.