Improving Access to Large Volumes of Online Data

Egemen Tanin Hanan Samet

Department of Computer Science
Center for Automation Research
Institute for Advanced Computer Studies
University of Maryland at College Park
{egemen, hjs}@cs.umd.edu

Abstract

The Internet has recently become the medium of interaction with large volumes of data. Enterprises in the public and private sectors made their databases available over the Internet. Working with such large volumes of online data is a challenging task. For efficient access to large data from remote locations we introduced APPOINT (an Approach for Peer-to-Peer Offloading the INTernet). In APPOINT, active clients of a client-server architecture act on the server's behalf and communicate with each other to transfer large volumes of online data more efficiently. In essence, a server is enabled to appoint alternatives and create a scalable collaborative virtual mirror from the active clients. Multiple parameters such as availability of clients and bandwidth information on clients are considered to decide on how to best forward a download request. APPOINT is 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 developed. Our experimental findings show that APPOINT can dramatically improve the performance of existing client-server based database systems.

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 and processing large volumes of data is a challenge. Many agency servers that make large volumes of data available over the Internet regularly slow to a crawl with a few large download requests. 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 interactively access a database at a remote location. Second, the browser along with an Internet-enabled database management system can be installed on the local architecture for more prolonged usages of the browser. In this case, the browser can still be utilized to view data from remote locations. However, in this second type of usage, the data that will be frequently used can be downloaded to the local database on demand, and subsequently accessed locally with high levels of continuous interactivity.

We want to help users that wish to manipulate large volumes of online data by developing APPOINT (an Approach for Peer-to-Peer Offloading the INTernet). APPOINT is a new peer-to-peer approach to provide users with the ability to transfer large volumes of data more efficiently by better utilizing the distributed network resources among active clients of an existing client-server architecture. We developed a library of functions, with a simple application programming interface (API), that can be easily plugged into an existing system. The results of this research address primarily the issues of the second type of usage for our browser. Although we are using a spatial data browser to test APPOINT, we believe that our approach is highly generalizable to other client-server applications.

2. Sample Application

SAND (denoting Spatial And Nonspatial Data) is a prototype spatial database system developed at the University of Maryland. The SAND Internet Browser [Samet et al., 2003], which is mainly a spatial data

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 for the applications that are used to interactively manipulate large volumes of online data. Figure 1 illustrates the output of an example query that finds all sites processing a certain chemical, arsenic, within a given distance of the border of Arkansas using the SAND Internet Browser and the SAND database system.

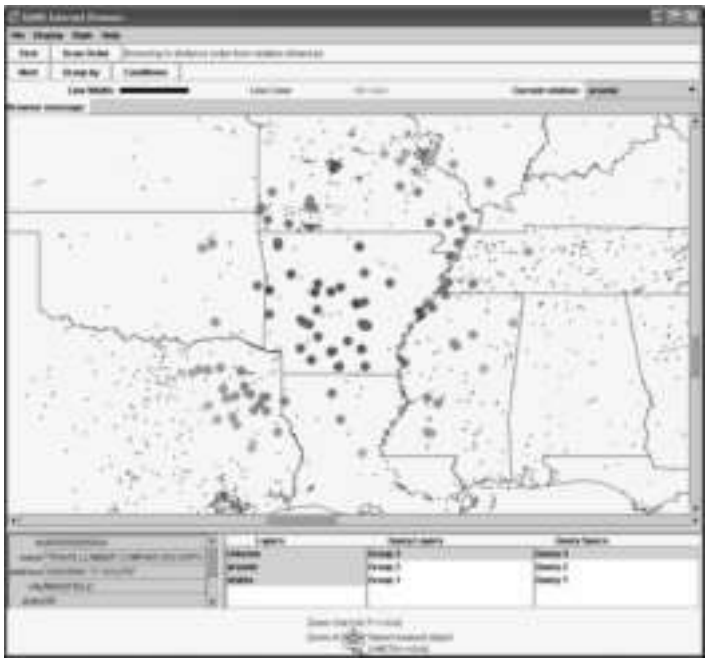

Figure 1: Sample output from the SAND Internet Browser looking at a data file, corresponding to the U.S. Environmental Protection Agency (EPA) regulated facilities that process arsenic. Large dark dots indicate the result of a query that looks for all arsenic sites within a given distance from Arkansas. Different color shades are used to indicate ranking order by the distance from a given point.

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. The queries are formed in an interactive fashion and then sent to the server, where the database management system is maintained, over the network. The results, e.g., vector data representing state boundaries, are sent back to the client for the browser to help the user to continue the formation of future queries. Also, the SAND Internet Browser along with the SAND spatial database system can be installed on the client side. In this 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 for longer periods of time with high levels of continuous interactivity. In time, users can build up their own local databases. We are using this local-application oriented latter case as our sample client-server application for APPOINT.

3. APPOINT

A few download requests to a large data file from some idle clients waiting to be served can slow the server to a crawl. The reason for this is that 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. We have built APPOINT as a peer-to-peer system to demonstrate our approach for improving the common client-server systems. A server still exists. It is the central source for the data and the decision center for the service. The environment still functions as a client-server environment under many circumstances. Meanwhile, APPOINT continuously maintains various types of information on the

clients. This includes, inventories of what each client has been downloading, their availabilities, bandwidths, etc. 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, have that data file from a previous download, and can take on the role of peers (hence, moving from a client-server scheme towards 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 of the requests. In this scheme, clients are used mainly for the purpose of sharing their networking resources rather than introducing new content 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 many other 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 normal 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 files 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 especially useful when the server's bandwidth is very limited. Nevertheless, the existence of a server assures that unpopular data is also available at all times (i.e., when we cannot find an appropriate client that currently has this data). The service depends on the existence of the server.

Backups and other maintenance activities are already being performed on the server and hence no extra administrative effort is needed for the active clients. If an active client 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 capabilities. The clients simply get on to or get off from a list of active clients on the server.

All of the operations are performed in a transparent fashion to the clients. Upon initial connection to the server, they can be queried as to whether or not they want to share their idle networking time and disk space (basically whether they want join to a community of sharing active clients). The rest of the operations follow transparently after the initial contact. APPOINT works on the application layer but not on lower layers. This achieves platform independence and easy deployment of the system. It is important to note that APPOINT is not a replacement for, but an addition to the current client-server architectures, and to achieve this we have developed a library of function calls that when placed in a client-server architecture starts the service. We also have developed comprehensive peer selection schemes that incorporate the bandwidth information of active clients, data size to be transferred, load on active clients, and availability of active clients to form a complete means of selecting the best active clients that can become efficient alternatives to the server.

With APPOINT we are defining a very simple application programming interface (API) that could be used easily within an existing client-server system. Instead of denial of service or a slow connection, the APPOINT API can be utilized to forward the service appropriately. The API that can be used on the server side is (i) start(serverPortNo), (ii) makeFileAvailable(file,fromLocation,boolean), and (iii) stop(). Similarly the API for the client side is (i) start(clientPortNo,serverPortNo,serverAddress), (ii) makeFileAvailable(file,fromLocation,boolean), (iii) receiveFile(file,toLocation), and (iv) stop().

The mechanism for a sample download operation is shown in Figure 2. As can be seen in the figure, there is an APPOINT component in all of the clients and there is also an APPOINT component in the server. Thus there are two types of APPOINT components. The client side component is responsible for creating a connection to the server, measuring bandwidth to the server and to other clients, relaying a request for a file, helping other clients with their downloads, receiving and compiling a file that has been requested,

and relaying information on the files that a client wants to share with others to the server. The server side component is responsible for accepting connections from clients, maintaining various types of information on clients, finding the best possible set of clients that may efficiently serve a download request when the request is received, and occasionally serving some of the data to the clients when no better alternatives are found.

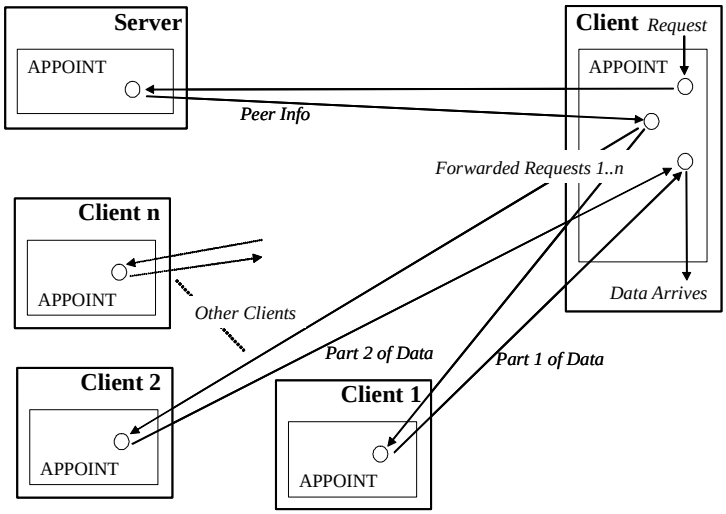

Figure 2: The download operation in APPOINT.

4. Experiment

We have built a simulation based evaluation environment for APPOINT. Our goal is to see how much, in terms of average download time, APPOINT can improve a client-server system. We used ns2, which is a popular simulator for networks, in tandem with the Georgia Tech Internetwork Topology Generator (GT-ITM) to create a simulation environment. A given set of download requests coming to a server in a random fashion during a 10 minute period formed the request list to be processed by the server. Hence, the average time for a download shows us how fast APPOINT or a simple client-server system can respond to a request in such an environment and scenario.

Figure 3 shows the results of our experiment. We ran this experiment to see the scalability of APPOINT with respect to the increasing number of clients. During this experiment, the average data file size was 25MBytes where all the data files were randomly generated between the minimum size of 1MBytes and the maximum size of 49MBytes. A client made an average of 5 data files available for downloads to the other clients where the size of this inventory of files on a client is chosen randomly between zero to 10. The total number of distinct data files was 20. The y-axis is the average time reported by the simulated environment for a client to complete a download in seconds under these input parameters and the x-axis is the increasing number of clients.

As expected, the client-server system showed a steep increase on average service time as the number of clients increased. APPOINT, on the other hand, scaled better with the increasing number of clients. Ideally, we should be able to accommodate any number of clients in APPOINT. The addition of more clients to the APPOINT community means new points of download for our data files. This should compensate for the extra load introduced by the requests of these new clients. Yet, the need to go to the peer selection routines in the server creates a gradual increase in the average download time as the number of clients increase. Hence, depending on the environment parameters, although APPOINT scales much better than a simple client-server architecture, it will also eventually slow to a crawl under a very large number of clients. This will be a different scale and type of problem. For a large number of clients

(e.g., 445), we can see a 3.73 fold difference between the client-server system and APPOINT. So, we can claim that the addition of APPOINT to a regular client-server application can dramatically improve the efficiency of the service. For smaller number of clients, we believe that using APPOINT may still be warranted. In particular, APPOINT was 1.40 times faster than the simple client-server system for a small number of clients (e.g., 45).

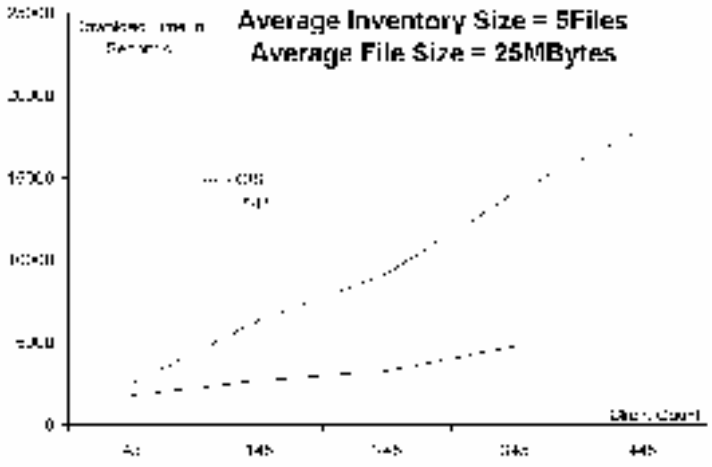

Figure 3: The comparison of a simple client-server system to APPOINT where the average number of files per client is 5 and the average data file size is 25MBytes. C/S denotes a client-server system and P2P denotes our peer-to-peer approach, APPOINT. The vertical lines on each data point show the standard deviation.

5. Related Work

Use of peer-to-peer systems on the Internet are gaining momentum [Oram, 2001]. Napster [Napster] and Gnutella [Gnutella] were the first and most well-known of these systems. Napster is a peer-to-peer system where the directory service is centralized on servers and people exchanged music files that they had on their disks. Gnutella is a more general 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 were 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, 2001], 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], made other resources, such as idle CPUs, work together over the Internet to solve large scale computational problems. Our goal is different from these approaches. We want to improve existing client-server systems in terms of performance by using the idle bandwidth among active clients with simple additions to the existing code. Hence, other issues like anonymity, decentralization, and persistence of storage were less important in our design decisions.

6. Future Work

Future work includes enhancing APPOINT so that it could be used for uploading data in a similar manner as we now use it for downloading data. We believe that for uploads, the active clients can again be utilized. Users can upload their data to a set of active clients other than the server if the server is busy or resides over a poor connection. Eventually the data can be propagated to the server. An additional API for the server side that may be needed to achieve this functionality is receivedFile(file,tolocation). Similarly, the additional API for the client side is sendFile(file,fromlocation).

The client side can send a file by using the sendFile method. The callback method of the server that will be invoked when a file is received from a client is receivedFile. This method can be used to trigger a set of other methods defined by the system programmer to handle the incoming data. Also, we may want to guarantee that a callback is called so that the user, who may not be online anymore, can always be notified (i.e., via email) by using this callback about the delivery of the file when needed. There are two possible choices to design APPOINT for uploads. One of them is to guarantee a delivery under all circumstances, which may be expensive in performance but more desirable for the users as it will be transparent to them. The other one is that there is a slight chance of incomplete delivery and a notification may be necessary for confirmations. The critical issue in the context of uploads is how to guarantee a successful delivery of a large data file at a certain probability when using a set of active but very dynamic clients while achieving higher levels of performance than a common client-server approach.

7. Conclusion

Interactive access to large volumes of online data is a challenge. In our work, we targeted improving service for users that manipulate large volumes of online data by developing a new system named APPOINT. APPOINT is designed to be a peer-to-peer approach for providing users with the ability to transfer large volumes of online data more efficiently. APPOINT utilizes the distributed network resources among active clients of a client-server architecture to achieve this goal. Our simulations show that we can dramatically improve the performance of current client-server systems. We have also observed the benefits of APPOINT even for a small number of clients that request data from a server.

We are currently using the SAND Internet Browser application as our sample client-server application for APPOINT. We plan to experiment with APPOINT on real users utilizing the SAND Internet Browser. APPOINT defines a very simple API that we are now using within an existing client-server system. We believe that other similar applications can also be easily improved using APPOINT. In the future, we also plan to incorporate uploads to APPOINT where a busy server can indirectly receive multiple data files from its clients.

Acknowledgments

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

References

[Anderson, 1996] R. Anderson. The Eternity service. In Proc. PRAGOCRYPT'96, pp. 242-252, CTU Publishing House, Prague, Czech Republic, 1996.

[Clarke et al., 2000] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, July 2000.

[Dabek et al., 2001] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In Proc. ACM SOSP'01, Banff, Canada, Oct. 2001.

[Dingledine et al., 2000] R. Dingledine, M. J. Freedman, and D. Molnar. The Free Haven project: Distributed anonymous storage service. In Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, July 2000.

[Gnutella] The Gnutella protocol specification. http://dss.clip2.com/GnutellaProtocol04.pdf.

[Kubiatowicz et al., 2000] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An architecture for global-scale persistent store. In Proc. ASPLOS'00, Cambridge, MA, Nov. 2000.

[Napster] Napster. http://www.napster.com/.

[Oram, 2001] A. Oram, ed. Peer-to-peer: Harnessing the power of disruptive technologies. O'Reilly and Assoc., CA, Mar. 2001.

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

[Rowstron and Druschel, 2001] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In Proc. ACM SOSP'01, Banff, Canada, Oct. 2001.

[Samet et al., 2003] H. Samet, H. Alborzi, F. Brabec, C. Esperanca, G. R. Hjaltason, F. Morgan, and E. Tanin. Use of the SAND Spatial Browser for Digital Government applications. Communications of the ACM, 46(1):63-66, January 2003.

[SETI] SETI@Home. http://setiathome.ssl.berkeley.edu/.