Remote Access to Large Spatial Databases *

Egemen Tanin
František Brabec
Hanan Samet

Computer Science Department
Center for Automation Research
Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742
{egemen,brabec,hjs}@umiacs.umd.edu
www.cs.umd.edu/{~egemen,~brabec,~hjs}

ABSTRACT

Enterprises in the public and private sectors have been making their large spatial data archives available over the Internet. However, interactive work with such large volumes of online spatial data is a challenging task. We propose two efficient approaches to remote access to large spatial data. First, we introduce a client-server architecture where the work is distributed between the server and the individual clients for spatial query evaluation, data visualization, and data management. We enable the minimization of the requirements for system resources on the client side while maximizing system responsiveness as well as the number of connections one server can handle concurrently. Second, for prolonged periods of access to large online data, we introduce APPOINT (an Approach for Peer-to-Peer Offloading the INTernet). This is a centralized peer-to-peer approach that helps Internet users transfer large volumes of online data efficiently. In APPOINT, active clients of the 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.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Client/server, Distributed applications, Distributed databases*; H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Performance, Management

Keywords

GIS, Internet, Client/server, Peer-to-peer

1. INTRODUCTION

In recent years, enterprises in the public and private sectors have provided access to large volumes of spatial data over the Internet. Interactive work with such large volumes of online spatial data is a challenging task. We have been developing an interactive browser for accessing spatial online databases: the SAND (Spatial and Non-spatial Data) Internet Browser. Users of this browser can interactively and visually manipulate spatial data remotely. Unfortunately, interactive remote access to spatial data slows to a crawl without proper data access mechanisms. We developed two separate methods for improving the system performance, together, form a dynamic network infrastructure that is highly scalable and provides a satisfactory user experience for interactions with large volumes of online spatial data.

The core functionality responsible for the actual database operations is performed by the server-based SAND system. SAND is a spatial database system developed at the University of Maryland [12]. The client-side 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.

SAND Internet Browser is Java-based, which makes it deployable across many platforms. In addition, since Java has often been installed on target computers beforehand, our clients can be deployed on these systems with little or no need for any additional software installation or customization. The system can start being utilized immediately without any prior setup which can be extremely beneficial in time-sensitive usage scenarios such as emergencies.

There are two ways to deploy SAND. First, any standard Web browser can be used to retrieve and run the client piece (SAND Internet Browser) as a Java application or an applet. This way, users across various platforms can continuously access large spatial data on a remote location with little or

*This work was supported in part by the National Science Foundation under Grants EIA-99-00268, EIA-99-01636, EAR-99-05844, IIS-00-86162, and EIA-00-91474.

GIS'02, November 8–9, 2002, McLean, Virginia, USA.

no need for any preceding software installation. The second option is to use a stand-alone SAND Internet Browser along with a locally-installed Internet-enabled database management system (server piece). In this case, the SAND Internet Browser can still be utilized to view data from remote locations. However, frequently accessed data can be downloaded to the local database on demand, and subsequently accessed locally. Power users can also upload large volumes of spatial data back to the remote server using this enhanced client.

We focused our efforts in two directions. We first aimed at developing a client-server architecture with 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 primarily addresses the issues of our first type of usage (i.e., as a remote browser application or an applet) for our browser and other similar applications. The second direction aims at helping users that wish to manipulate large volumes of online data for prolonged periods. We have developed a centralized peer-to-peer approach to provide the users with the ability to transfer large volumes of data (i.e., whole data sets to the local database) more efficiently by better utilizing the distributed network resources among active clients of a client-server architecture. We call this architecture APPOINT — Approach for Peer-to-Peer Offloading the INTernet. The results of this research addresses primarily the issues of the second type of usage for our SAND Internet Browser (i.e., as a stand-alone application).

The rest of this paper is organized as follows. Section 2 describes our client-server approach in more detail. Section 3 focuses on APPOINT, our peer-to-peer approach. Section 4 discusses our work in relation to existing work. Section 5 outlines a sample SAND Internet Browser scenario for both of our remote access approaches. Section 6 contains concluding remarks as well as future research directions.

2. THE CLIENT-SERVER APPROACH

Traditionally, Geographic Information Systems (GIS) such as ArcInfo from ESRI [2] and many spatial databases are designed to be stand-alone products. The spatial database is kept on the same computer or local area network from where it is visualized and queried. This architecture allows for instantaneous transfer of large amounts of data between the spatial database and the visualization module so that it is perfectly reasonable to use large-bandwidth protocols for communication between them. There are however many applications where a more distributed approach is desirable. In these cases, the database is maintained in one location while users need to work with it from possibly distant sites over the network (e.g., the Internet). These connections can be far slower and less reliable than local area networks and thus it is desirable to limit the data flow between the database (server) and the visualization unit (client) in order to get a timely response from the system.

Our client-server approach (Figure 1) allows the actual database engine to be run in a central location maintained by spatial database experts, while end users acquire a Java-based client component that provides them with a gateway into the SAND spatial database engine.

Our client is more than a simple image viewer. Instead, it operates on vector data allowing the client to execute many operations such as zooming or locational queries locally. In

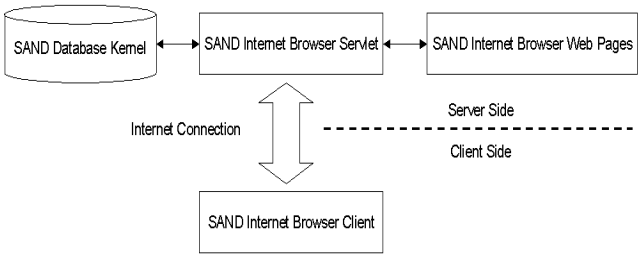

Figure 1: SAND Internet Browser — Client-Server architecture.

essence, a simple spatial database engine is run on the client. This database keeps a copy of a subset of the whole database whose full version is maintained on the server. This is a concept similar to 'caching'. In our case, the client acts as a lightweight server in that given data, it evaluates queries and provides the visualization module with objects to be displayed. It initiates communication with the server only in cases where it does not have enough data stored locally.

Since the locally run database is only updated when additional or newer data is needed, our architecture allows the system to minimize the network traffic between the client and the server when executing the most common user-side operations such as zooming and panning. In fact, as long as the user explores one region at a time (i.e., he or she is not panning all over the database), no additional data needs to be retrieved after the initial population of the client-side database. This makes the system much more responsive than the Web mapping services. Due to the complexity of evaluating arbitrary queries (i.e., more complex queries than window queries that are needed for database visualization), we do not perform user-specified queries on the client. All user queries are still evaluated on the server side and the results are downloaded onto the client for display. However, assuming that the queries are selective enough (i.e., there are far fewer elements returned from the query than the number of elements in the database), the response delay is usually within reasonable limits.

2.1 Client-Server Communication

As mentioned above, the SAND Internet Browser is a client piece of the remotely accessible spatial database server built around the SAND kernel. In order to communicate with the server, whose application programming interface (API) is a Tcl-based scripting language, a servlet specifically designed to interface the SAND Internet Browser with the SAND kernel is required on the server side. This servlet listens on a given port of the server for incoming requests from the client. It translates these requests into the SAND-Tcl language. Next, it transmits these SAND-Tcl commands or scripts to the SAND kernel. After results are provided by the kernel, the servlet fetches and processes them, and then sends those results back to the originating client.

Once the Java servlet is launched, it waits for a client to initiate a connection. It handles both requests for the actual client Java code (needed when the client is run as an applet) and the SAND traffic. When the client piece is launched, it connects back to the SAND servlet, the communication is driven by the client piece; the server only responds to the client's queries. The client initiates a transaction by

sending a query. The Java servlet parses the query and creates a corresponding SAND-Tcl expression or script in the SAND kernel's native format. It is then sent to the kernel for evaluation or execution. The kernel's response naturally depends on the query and can be a boolean value, a number or a string representing a value (e.g., a default color) or, a whole tuple (e.g., in response to a nearest tuple query). If a script was sent to the kernel (e.g., requesting all the tuples matching some criteria), then an arbitrary amount of data can be returned by the SAND server. In this case, the data is first compressed before it is sent over the network to the client. The data stream gets decompressed at the client before the results are parsed.

Notice, that if another spatial database was to be used instead of the SAND kernel, then only a simple modification to the servlet would need to be made in order for the SAND Internet Browser to function properly. In particular, the queries sent by the client would need to be recoded into another query language which is native to this different spatial database. The format of the protocol used for communication between the servlet and the client is unaffected.

3. THE PEER-TO-PEER APPROACH

Many users may want to work on a complete spatial data set for a prolonged period of time. In this case, making an initial investment of downloading the whole data set may be needed to guarantee a satisfactory session. Unfortunately, spatial data tends to be large. A few download requests to a large data set from a set of idle clients waiting to be served can slow the server to a crawl. This is due to the fact 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 built APPOINT as a centralized peer-to-peer system to demonstrate our approach for improving the common client-server systems. A server still exists. There is a central source for the data and a decision mechanism for the service. The environment still functions as a client-server environment under many circumstances. Yet, unlike many common client-server environments, APPOINT maintains more information about the clients. This includes, inventories of what each client downloads, their availabilities, 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 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 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 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 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 server assures that unpopular data is also available at all times. The service depends on the availability of the server. The server is now more resilient to congestion as the service is more scalable.

Backups and other maintenance activities are already being performed on the 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 server if the server is busy or resides in a distant location. Eventually the data is propagated to 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. 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. APPOINT is not a replacement but an addition to the current client-server architectures. We developed a library of function calls that when placed in a client-server architecture starts the service. We are developing advanced peer selection schemes that incorporate the location of active clients, bandwidth among 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 clients that can become efficient alternatives 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 or a slow connection, this API can be utilized to forward the service appropriately. The API for the server side is:

```
start(serverPortNo)
makeFileAvailable(file,location,boolean)
callback receivedFile(file,location)
callback errorReceivingFile(file,location,error)
stop()
```

Similarly the API for the client side is:

```
start(clientPortNo,serverPortNo,serverAddress)
makeFileAvailable(file,location,boolean)
receiveFile(file,location)
sendFile(file,location)
stop()
```

The server, after starting the APPOINT service, can make all of the data files available to the clients by using the `makeFileAvailable` method. This will enable APPOINT to treat the server as one of the peers.

The two callback methods of the server are invoked when a file is received from a client, or when an error is encountered while receiving a file from a client. APPOINT guar-

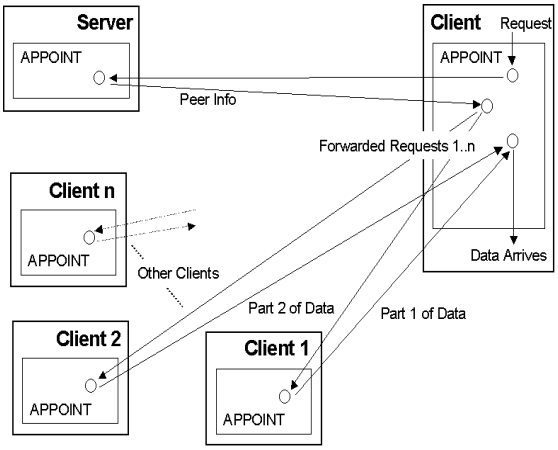

Figure 2: The localization operation in APPOINT.

antees that at least one of the callbacks will be called so that the user (who may not be online anymore) can always be notified (i.e., via email). Clients localizing large data files can make these files available to the public by using the `makeFileAvailable` method on the client side.

For example, in our SAND Internet Browser, we have the localization of spatial data as a function that can be chosen from our menus. This functionality enables users to download data sets completely to their local disks before starting their queries or analysis. In our implementation, we have calls to the APPOINT service both on the client and the server sides as mentioned above. Hence, when a localization request comes to the SAND Internet Browser, the browser leaves the decisions to optimally find and localize a data set to the APPOINT service. Our server also makes its data files available over APPOINT. The mechanism for the localization operation is shown with more details from the APPOINT protocols in Figure 2. The upload operation is performed in a similar fashion.

4. RELATED WORK

There has been a substantial amount of research on remote access to spatial data. One specific approach has been adopted by numerous Web-based mapping services (MapQuest [5], MapsOnUs [6], etc.). The goal in this approach is to enable remote users, typically only equipped with standard Web browsers, to access the company's spatial database server and retrieve information in the form of pictorial maps from them. The solution presented by most of these vendors is based on performing all the calculations on the server side and transferring only bitmaps that represent results of user queries and commands. Although the advantage of this solution is the minimization of both hardware and software resources on the client site, the resulting product has severe limitations in terms of available functionality and response time (each user action results in a new bitmap being transferred to the client).

Work described in [9] examines a client-server architecture for viewing large images that operates over a low-bandwidth network connection. It presents a technique based on wavelet transformations that allows the minimization of the amount of data needed to be transferred over the network between the server and the client. In this case, while the server holds the full representation of the large image, only a limited amount of data needs to be transferred to the client to enable it to display a currently requested view into the image. On the client side, the image is reconstructed into a pyramid representation to speed up zooming and panning operations. Both the client and the server keep a common mask that indicates what parts of the image are available on the client and what needs to be requested. This also allows dropping unnecessary parts of the image from the main memory on the server.

Other related work has been reported in [16] where a client-server architecture is described that is designed to provide end users with access to a server. It is assumed that this data server manages vast databases that are impractical to be stored on individual clients. This work blends raster data management (stored in pyramids [22]) with vector data stored in quadtrees [19, 20].

For our peer-to-peer transfer approach (APPOINT), Napster is the forefather where a directory service is centralized on a server and users exchange music files that they have stored on their local disks. Our application domain, where the data is already freely available to the public, forms a prime candidate for such a peer-to-peer approach. Gnutella is a pure (decentralized) peer-to-peer file exchange system. Unfortunately, it suffers from scalability issues, i.e., floods of messages between peers in order to map connectivity in the system are required. Other systems followed these popular systems, each addressing a different flavor of sharing over the Internet. Many peer-to-peer storage systems have also recently emerged. PAST [18], Eternity Service [7], CFS [10], and OceanStore [15] are some peer-to-peer storage systems. Some of these systems have focused on anonymity while others have focused on persistence of storage. Also, other approaches, like SETI@Home [21], 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. With APPOINT, 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. Confirming the authenticity of the indirectly delivered data sets is not yet addressed with APPOINT. We want to expand our research, in the future, to address this issue.

From our perspective, although APPOINT employs some of the techniques used in peer-to-peer systems, it is also closely related to current Web caching architectures. Squirrel [13] forms the middle ground. It 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 topic in the realm of server/proxy level caching [8, 11, 14, 17]. 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 arrives at a 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 down-

load and upload operations. Cache misses are especially common in the type of large data-based services on which we are working. 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. Finally, none of the Web caching methods address the symmetric issue of large data uploads.

5. A SAMPLE APPLICATION

FedStats [1] is an online source that enables ordinary citizens access to official statistics of numerous federal agencies without knowing in advance which agency produced them. We are using a FedStats data set as a testbed for our work. Our goal is to provide more power to the users of FedStats by utilizing the SAND Internet Browser. As an example, we looked at two data files corresponding to Environmental Protection Agency (EPA)-regulated facilities that have chlorine and arsenic, respectively. For each file, we had the following information available: EPA-ID, name, street, city, state, zip code, latitude, longitude, followed by flags to indicate if that facility is in the following EPA programs: Hazardous Waste, Wastewater Discharge, Air Emissions, Abandoned Toxic Waste Dump, and Active Toxic Release.

We put this data into a SAND relation where the spatial attribute 'location' corresponds to the latitude and longitude. Some queries that can be handled with our system on this data include:

1. Find all EPA-regulated facilities that have arsenic and participate in the Air Emissions program, and:
   (a) Lie in Georgia to Illinois, alphabetically.
   (b) Lie within Arkansas or 30 miles within its border.
   (c) Lie within 30 miles of the border of Arkansas (i.e., both sides of the border).
2. For each EPA-regulated facility that has arsenic, find all EPA-regulated facilities that have chlorine and:
   (a) That are closer to it than to any other EPA-regulated facility that has arsenic.
   (b) That participate in the Air Emissions program and are closer to it than to any other EPA-regulated facility which has arsenic. In order to avoid reporting a particular facility more than once, we use our 'group by EPA-ID' mechanism.

Figure 3 illustrates the output of an example query that finds all arsenic sites within a given distance of the border of Arkansas. The sites are obtained in an incremental manner with respect to a given point. This ordering is shown by using different color shades.

With this example data, it is possible to work with the SAND Internet Browser online as an applet (connecting to a remote server) or after localizing the data and then opening it locally. In the first case, for each action taken, the client-server architecture will decide what to ask for from the server. In the latter case, the browser will use the peer-to-peer APPOINT architecture for first localizing the data.

6. CONCLUDING REMARKS

An overview of our efforts in providing remote access to large spatial data has been given. We have outlined our approaches and introduced their individual elements. Our client-server approach improves the system performance by using efficient caching methods when a remote server is accessed from thin-clients. APPOINT forms an alternative approach that improves performance under an existing client-server system by using idle client resources when individual users want work on a data set for longer periods of time using their client computers.

For the future, we envision development of new efficient algorithms that will support large online data transfers within our peer-to-peer approach using multiple peers simultaneously. We assume that a peer (client) can become unavailable at any anytime and hence provisions need to be in place to handle such a situation. To address this, we will augment our methods to include efficient dynamic updates. Upon completion of this step of our work, we also plan to run comprehensive performance studies on our methods.

Another issue is how to access data from different sources in different formats. In order to access multiple data sources in real time, it is desirable to look for a mechanism that would support data exchange by design. The XML protocol [3] has emerged to become virtually a standard for describing and communicating arbitrary data. GML [4] is an XML variant that is becoming increasingly popular for exchange of geographical data. We are currently working on making SAND XML-compatible so that the user can instantly retrieve spatial data provided by various agencies in the GML format via their Web services and then explore, query, or process this data further within the SAND framework. This will turn the SAND system into a universal tool for accessing any spatial data set as it will be deployable on most platforms, work efficiently given large amounts of data, be able to tap any GML-enabled data source, and provide an easy to use graphical user interface. This will also convert the SAND system from a research-oriented prototype into a product that could be used by end users for accessing, viewing, and analyzing their data efficiently and with minimum effort.

7. REFERENCES

[1] Fedstats: The gateway to statistics from over 100 U.S. federal agencies. `http://www.fedstats.gov/`, 2001.

[2] Arcinfo: Scalable system of software for geographic data creation, management, integration, analysis, and dissemination. `http://www.esri.com/software/arcgis/arcinfo/index.html`, 2002.

[3] Extensible markup language (xml). `http://www.w3.org/XML/`, 2002.

[4] Geography markup language (gml) 2.0. `http://opengis.net/gml/01-029/GML2.html`, 2002.

[5] Mapquest: Consumer-focused interactive mapping site on the web. `http://www.mapquest.com`, 2002.

[6] Mapsonus: Suite of online geographic services. `http://www.mapsonus.com`, 2002.

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

[8] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions:

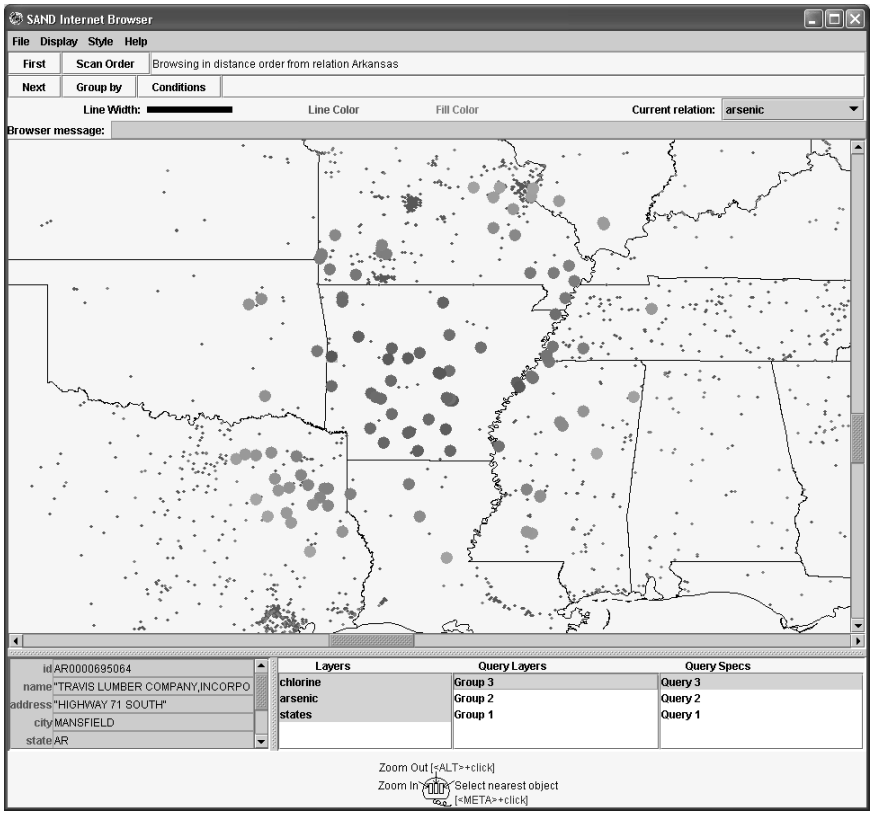

Figure 3: Sample output from the SAND Internet Browser — Large dark dots indicate the result of a query that looks for all arsenic sites within a given distance from Arkansas. Different color shades are used to indicate ranking order by the distance from a given point.

Evidence and implications. In *Proceedings of the IEEE Infocom'99*, pages 126–134, New York, NY, March 1999.

[9] E. Chang, C. Yap, and T. Yen. Realtime visualization of large images over a thinwire. In R. Yagel and H. Hagen, editors, *Proceedings IEEE Visualization'97 (Late Breaking Hot Topics)*, pages 45–48, Phoenix, AZ, October 1997.

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

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

[12] C. Esperança and H. Samet. Experience with SAND/Tcl: a scripting tool for spatial databases. *Journal of Visual Languages and Computing*, 13(2):229–255, April 2002.

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

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

[15] 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 *Proceedings of the ACM ASPLOS'00*, pages 190–201, Cambridge, MA, November 2000.

[16] M. Potmesil. Maps alive: viewing geospatial information on the WWW. *Computer Networks and ISDN Systems*, 29(8–13):1327–1342, September 1997. Also *Hyper Proceedings of the 6th International World Wide Web Conference*, Santa Clara, CA, April 1997.

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

[18] A. Rowstron and P. Druschel. 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, October 2001.

[19] H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS.* Addison-Wesley, Reading, MA, 1990.

[20] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, MA, 1990.

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

[22] L. J. Williams. Pyramidal parametrics. *Computer Graphics*, 17(3):1–11, July 1983. Also *Proceedings of the SIGGRAPH'83 Conference*, Detroit, July 1983.