Chapter 1
Hierarchical Infrastructure for Internet Mapping Services

František Brabec and Hanan Samet[⋆]

Abstract For years, the access to internet-based public mapping services provided by vendors such as MapQuest or MapsOnUs has changed little. The mapping service would generate maps of the viewed areas in raster format and transfer them in the form of images embedded in web pages to remote users. This approach is suboptimal for users who plan to explore a given area in more detail as the same data may be sent to the users repeatedly. In mid 2005, Google Maps and MS Virtual Earth improved upon this approach by dividing the images into smaller tiles which allows many of them to be reused in subsequent panning. This increases performance of such mapping systems substantially. In both cases, however, the client only has access to data converted in its raster format which prevents it from querying or re-processing the data locally. We investigate this opportunity for further improvement in providing the client with map data in vector format so that it can perform some operations locally without accessing the server. We focus on finding strategies for distributing of work between the server, clients, and possibly other entities introduced into the model for query evaluation and data management. We address issues of scalability for clients that have only limited access to system resources (e.g., a Java applet). We compare performance of the vector-based system with raster-based systems, both traditional (e.g., MapQuest) and tiled methods (e.g., Google Maps) for a set of common basic operations consisting of fine and fast scrolling and zooming (both in and out).

František Brabec
Department of Computer Science, University of Maryland, College Park, Maryland 20742
e-mail: `brabec@cs.umd.edu`

Hanan Samet
Department of Computer Science, University of Maryland, College Park, Maryland 20742
e-mail: `hjs@cs.umd.edu`

⋆ The support of the National Science Foundation under Grants EIA-00-91474, CCF-0515241, and IIS-0713501; Microsoft Research; and the University of Maryland General Research Board is gratefully acknowledged.

1.1 Introduction

Technological advances in recent years have opened ways for easier creation of spatial data. Vast amounts of data are collected daily by both governmental institutions (e.g., USGS, NASA) and commercial entities (e.g., IKONOS) for a wide range of scientific applications (e.g., [18]).

The motivation is the increased popularity and affordability across the spectrum of collection methods, ranging from personal GPS units to satellite systems. Many collection methods such as satellite systems produce data in raster format. Often, such raster data is analyzed by researchers directly, while at other times such data is used to produce a final dataset in vector format. With rapidly increasing data supplies, more applications for the data are being developed that interest a wider consumer base. The increasing popularity of spatial data viewers and query tools with end users introduces a requirement for methods to allow these users to access this data for viewing and querying instantly and without much effort. Our work focuses on providing remote access to vector-based spatial data, rather than raster data.

Traditionally, common spatial databases and Geographic Information Systems (GIS) such as ESRI's ArcInfo are designed to be stand-alone products. The spatial database is kept on the same computer or local area network from which it is visualized and queried. 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 places over the network (e.g., the public internet). A common approach for providing access to remote spatial databases adopted by numerous web-based mapping service vendors (e.g., MapQuest [8]) performs all the operations on the server side, and then transfers only bitmaps that represent results of user queries and commands. Although this solution only requires minimal hardware and software resources on the client site, the resulting product is severely limited in the available functionality and response time (each user action results in a new bitmap being transferred to the client). Naturally, the drawbacks of this traditional approach have been identified and work has started to improve the performance of remote spatial access using both raster [1] and vector [19] approaches. Similar issues were addressed in a component-based WebGIS [12] tool by adding a spatial caching framework.

Providing efficient data flow between a given spatial server and individual clients is not the only problem that needs addressing. In many scenarios, data originates from multiple providers and the information they offer needs to be aggregated before being presented to the end user. Similarly, multiple spatial servers may be involved for redundancy or load balancing. Such topics have been explored in other work, when the providers' hosting environment remains stable [20] and for more dynamic peer-to-peer arrangements [11].

In our research, we explore new ways of allowing visualization of both spatial and nonspatial data stored in a central server database on a simple client connected to this server by a possibly slow and unreliable connection.

We develop a new vector-based client-server approach as a response to some of these drawbacks of traditional solutions. Our system aims to partition the workload between the client and the server in such a manner that the user's experience with the system is interactive, with minimal delay between the user action and appropriate response. We consider scenarios where bringing in auxiliary servers would improve the performance of the system. The design works around potential bottlenecks for the information transfer such as the limited network bandwidth or resources available on the client computer. To support multiple concurrent clients, limited resources on the server must also be considered. We will see that the performance of our vector approach is comparable and at times better than the latest raster-based methods.

The rest of the paper is organized as follows. Section 1.2 reviews existing commonly used methods for remotely accessing spatial databases. Section 1.3 discusses our architecture based on pure client-server approach. Given a client that communicates directly to a server, we examine different deployment options and describe several methods that improve the performance that can be achieved in this environment. Section 1.4 extends the basic client-server approach by adding auxiliary servers. Such servers can be used as temporary data storage between the client and the server. We present typical deployment scenarios when this would be beneficial, as well as present methods for using this arrangement to further speed up its performance. Section 1.5 combines all the different design options and speed-up methods together, performs evaluations, and discusses how to choose the optimal deployment method for given specific usage scenarios. Section 1.6 shows results of experiments comparing performance of our method and existing established raster-based remote access methods. Finally, Section 1.7 draws some conclusions and proposes topics for further research.

1.2 Internet Mapping Services

Many vendors that provide access to maps over the web utilize an approach where server-generated bitmaps are sent to the web browser client for viewing. The typical example of providers of such services are vendors such as MapQuest [8] for street maps based on addresses; or TopoZone [7] for topographical maps. Their approach is simple, the server receives a location description (e.g., a street address, name of a place, etc), it queries its spatial database, retrieves a map, converts it into a bitmap image and sends it back to the user (their browser). The map retrieved from the spatial database may be in vector (MapQuest) or raster (TopoZone) format. In either case, it gets rasterized or subsampled respectively before sending the data over the network to user's browser.

This approach requires very little support from the client site, typically just a web-browser equipped computer or network appliance. The drawback of this solution is that it quickly reaches its usability limitations when more serious work is attempted. Such poorly supported operations include even basic zooming in or out or panning not to mention running queries. In particular, actions such as zooming or panning are very cumbersome with performance bordering unacceptable for many users as the response time is determined by the amount of data that needs to be transferred every time a new view is requested. Other operations such as querying the database beyond displaying all objects within a certain rectangle are not supported at all.

An interesting enhanced raster-based design has recently been presented by Google [1] and Microsoft [3]. Similar to MapQuest, Google Maps and Microsoft Virtual Earth (and its predecessor TerraServer [10]) services are raster-based as is also NASA's World Wind [5] which besides working with NASA's own data it also makes use of data from TerraServer. However, these services do not send a single image covering the whole viewable area every time there is a need for an update. Instead, the viewable map is divided into a grid of smaller image cells. When a panning operation is executed, there is no need to download a new image that represents the whole viewable area. Only cells covering the area that just became visible need to be downloaded, others are reused by simply moving them on the screen.

As an alternative to these raster-based systems, we consider the SAND Internet Browser [17] — a Java application that represents the client piece of our vector-based client-server solution for facilitation of remote access to spatial databases.

1.3 Direct Server Access

Traditionally, a client-server computing paradigm only involves two computers — the client and the server (obviously ignoring computers and devices in between the two that simply route or shape the traffic between them, such as routers, firewalls, etc). We examine such a scenario as well as scenarios that involve other auxiliary servers.

1.3.1 Pure Client-Server Design

The simplest and most common design for the client-server architecture makes individual tasks such as data management, image rendering, and query evaluation the responsibility of either the client or the server. When the spatial database application is implemented in this manner, the server handles all the data management and query evaluation. The client only facilitates data visualization while maintaining connectivity to the server. In this scenario, the client simply translates user input into queries and transmits them to the server. It can also receive data sent by the server and visualize it. There is no data storage or processing on the client beyond these basic functions. Note that this design corresponds to the way in which many popular web-based mapping services such as MapQuest operate.

This approach's advantage is that most users can utilize the service with the resources that they already have — that is, a networked machine with a web browser. Users do not need to install or set up any additional hardware or software. However, this approach's main drawback is that clients need to communicate with the server each time users request even the simplest operation. This can slow down users experience significantly if the network throughput and latency are a limiting factor or if the server is heavily loaded.

1.3.2 Memory-Based Caching in the Client

The first method that improves upon the basic design is one where the client utilizes some of its own main memory to store (cache) some of the data in the central database. This allows the client in some cases to rely on its own data repository to handle some of the user's requests thus cutting back on the network utilization and improving the system's responsiveness. Naturally, the spatial data stored on the client must be spatially indexed for fast access. Note that in this approach it is no longer possible to use the standard web browser as a mere image viewer. In particular, custom code needs to be loaded onto the client to facilitate the operations to be performed there. The Java environment has emerged in the past years as a platform of choice for most types of lightweight cross-platform applications. The maximum amount of data to be stored on the client to optimize the overall performance depends on the client's available resources. The rationale for this design is for the client to fetch the requested data via fast memory-only operations whenever possible. This is more efficient than retrieving the same data over the network from the central server.

Operations performed by the mapping system are primarily client-driven, i.e., any operation performed on either the client or the server is in response to some user-generated input. To minimize the amount of data that needs to be transferred from the server to the client in response to each event on the client side, various techniques were developed and implemented in the form of the SAND Internet Browser. To keep the amount of traffic between the client and the server low, we cache some data on the client in case the user requests another operation on data in the same area. We store the data in their original vector format rather than the resulting bitmaps so that the client is able to generate new views and process some types of queries locally without having to request additional data from the server.

1.3.3 Internal Spatial Data Structures

The spatial data is stored on the client using a PMR quadtree [13] spatial data structure. This structure subdivides the plane into quadrants such that if an object is inserted into a certain quadrant, then if the quadrant already contains more than a predefined threshold of other objects, then the quadrant is split into its four children once and only once and the objects are reinserted into the children. Thus, the objects are always stored in the leaf nodes of this quadtree. We establish and maintain the maximum amount of data that can be cached on the client in order not to overwhelm or crash the client platform. Each leaf node of the PMR quadtree also contains a time stamp indicating when it was accessed (displayed) last. Together with the PMR quadtree containing the spatial data, we also maintain pointers to all of the PMR quadtree leaf nodes using a variant of a binary heap data structure. The key for this tree is the time stamp stored in the PMR quadtree leaf nodes. This is shown in Figure 1.1.

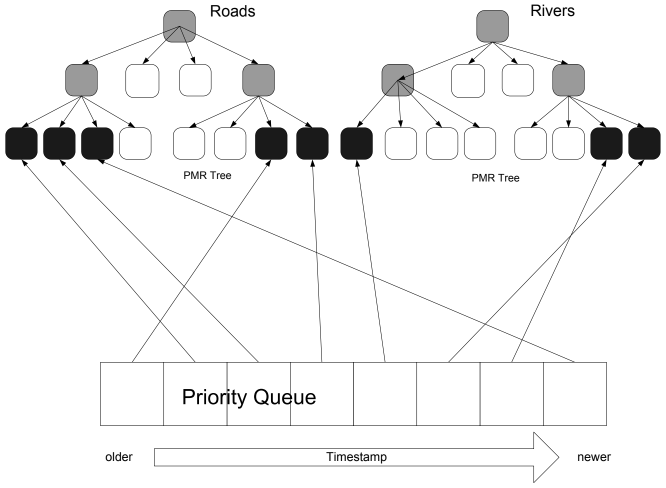


Figure 1.1: Individual spatial data layers are stored in separate PMR quadtrees. A priority queue shared by all of them maintains ordering of all the PMR leaves for all the PMR quadtrees based on the time of their last viewing.

This structure enables quick insertions, deletions and locating the pointer representing the PMR node with the oldest time stamp. This arrangement facilitates our caching mechanism. When we need to make more memory available for additional data, we use the least-recently-used (LRU) caching mechanism to delete as many PMR leaf nodes linked from the top of the binary heap data structure as necessary. If all four children of some internal

PMR quadtree node are removed, then the quadtree automatically collapses and the internal node becomes an empty leaf node. A flag in each node indicates whether the node represents an area that is actually empty (valid node) or whether the node is empty because its elements are not available in the memory (e.g., page fault, invalid node).

Note that using this mechanism, the entire quadrant has to be contained in the memory for its node to be valid. This may be too inefficient as if we continuously work with only part of the quadrant and have no need to load the rest of the quadrant, then the node would never be marked as valid and the data from the part in which we are interested would be reloaded over and over. To prevent this, we add another field in each node indicating which part of it is actually valid. Thus, if we loaded data for only part of the quadrant, we mark the quadrant as valid but indicate which part of it is actually valid (i.e., the intersection of the quadrant and the query window). The next time we need to access data from this quadrant, if the area that we need falls completely within the valid area of the node, then we do not need to load any additional data. If the area that we need is not fully enclosed by the valid area, then we load the missing part and increase the valid area of the node accordingly.

A typical dataset would contain several tables representing different layers of the map. While each layer is stored in a separate PMR quadtree, there is only a single binary heap data structure for all the layers combined. This way, when a user stops working with one of the layers, its data will be automatically and gradually removed from the cache and will be replaced with the data needed currently.

As the user explores the content of the database using a graphical viewer, s/he is basically retrieving all the objects stored in the database that overlap the current viewing window. When the content of the spatial structure overlapping a certain query object needs to be drawn, a tree traversal is performed to find all the objects in the PMR quadtree that overlap the query object. At times, we find that the internal nodes are "invalid" which means that either they were not loaded yet or they were previously removed by the memory management process when they were not used for some time. In such a case, the data needs to be reloaded.

The algorithm contains two steps. In the first step, the system finds out what areas need to be loaded from the server and builds a collection of rectangles that represent this area.

In the second step, the algorithm takes the list of rectangles returned by the first step and loads all the data from the server that lie within the area defined by this collection of rectangles. Next, for each rectangle loaded, it adjusts the corresponding PMR node status.

Now, when we need to display all data that overlaps a given window w, we can look at not just the valid/invalid identifier of each PMR node that overlaps w, but instead we can also check the *validSubarea* field of the invalid nodes. If the intersection of the window w with the PMR block is fully contained in the node's *validSubarea*, then we know that all the necessary data for this window query is already in the database, even if the PMR node is not loaded fully. When the drawing function is called, it already knows that all the data is already loaded in the memory and it simply steps through the overlapping PMR nodes and displays their contents.

An obvious limitation of the memory-only approach is the maximum amount of space that can be utilized for local data storage. This approach is the only one available when the client runs on a platform that has no secondary memory (e.g., disks) available. Such an environment is usually present on smaller handheld devices or on Java applet-based viewers. Various SQL-based DBMSs exist for many platforms that can be used to facilitate local caching using available disk space.

1.4 Utilizing Auxiliary Servers

Development of internet technologies has introduced various methods for utilization of additional servers to improve performance for end users who connect to external servers. One of the first and most popular methods is caching. Caching can be implemented directly within the end user's browser, or it can also be implemented within the user's network, on the gateway (proxy) between the network and the outside internet. In the latter case, the same cache can be shared among several users.

Obviously, the rationale for introducing these proxy servers between the client and the internet is that the responsiveness of the proxy server with respect to the end user's browser is much higher than if the data was requested directly from the original host. This is due to the higher network speed between the client and the proxy server compared to the network speed between the client and the original host. Another factor can possibly be the lower load and higher responsiveness of the proxy server since it only handles traffic for a few users and therefore can process requests more efficiently.

An example deployment of a proxy server is an emergency situation illustrated in Figure 1.2. There, multiple first responders equipped with handheld devices link with a mobile communication van or similar vehicle. This vehicle is equipped with a wireless router as well as with satellite or similar communication technology and facilitates connectivity with the central computing facilities.

1.4.1 Static Proxy

In some cases, the main spatial server provider and the individual users of this database are from within the same organization or these organizations

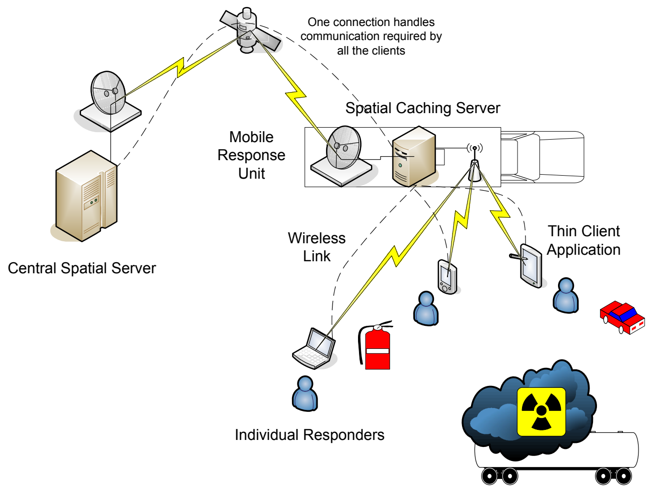


Figure 1.2: Emergency response service deployed a mobile unit (e.g., a mobile van) in support of the operations. This unit can cover the area with a fast wireless network access and provide a proxy service for spatial operations. Individual responders can utilize the applications on their mobile devices more efficiently.

collaborate closely. If this is the case and the spatial data is rather static (i.e., updates in the database are not performed frequently), it may be feasible to execute a one-time step of copying all the spatial data stored in the main spatial database onto the auxiliary database running on the proxy server. In such a scenario, the auxiliary database needs to be preloaded with the spatial data from the central SAND server when the system is being installed as well as possibly periodically after that.[2] The frequency would depend on how often the data on the central server changes. This approach is especially effective if updates on the central spatial server are performed in regular intervals rather than dynamically. For instance, a new data set may be released once a month or once a year instead of applying partial updates continuously.

Since the complete valid map resides on the proxy server, there is no need for the client to ever connect to the central spatial server for window queries. There is also no need for the proxy server to talk to the spatial server, to receive updates or for any other reason. Therefore, the only traffic generated by this scheme involves the SAND Internet Browser clients communicating

[2] This arrangement is similar to setting up a mirror server. The difference is that a mirror server is typically a copy of the primary server and can provide any functionality that the primary server does. In this case, the proxy only stores spatial data of the background map and facilitates window queries. The central server is still used to evaluate complex custom queries as initiated by the user.

with both the central spatial server (e.g., SAND server) and the auxiliary proxy server.

1.4.2 Dynamic Proxy

In other cases, the amount of data stored on the central server would overwhelm even a normal server-level machine. Or, the data on the central server gets updated continuously and any information stored on the server may potentially be valid for only a short period of time. In such scenarios, preloading the proxy server with all the spatial data from the main spatial server is not possible and/or useful. For this situation, we have developed a design that involves deploying the proxy server with no data preloaded on it. As the individual clients start working with the data, they still go directly to the central spatial server to get results for custom queries and to the proxy server to get results of window queries. This time however, the necessary data may or may not be available on the proxy server. If the data is available, it is sent back to the client immediately. If the data is not available, then the proxy connects to the central spatial server, retrieves the necessary data and stores it in its database. Once this is finished, the proxy server evaluates the window query locally. As the data was just loaded, the server retrieves all the data successfully and sends it back to the client. The layout of this scenario is illustrated in Figure 1.3.

Since the dynamic proxy loads the data from the central spatial server on as-needed basis, it is not a problem if some data is not available locally. The proxy can utilize this to drop data when necessary, e.g., to keep the amount of data stored locally under a prescribed limit or to ensure that the data served is not older than a certain predetermined age. This approach can be used as described if the data on the server does not change (e.g., a street map). If the data on the server is updated frequently, then the server needs to notify its clients that a certain part of the database was updated. In response, the clients drop the corresponding data from their cache and will reload it the next time a user requests it.

1.4.3 Implementation Details

The SAND Internet Browser running on clients is implemented in Java and its connection with the external servers is facilitated via Java Database Connectors (JDBC) modules provided by the respective database vendors.

The SAND Proxy, the implementation of the proxy server outlined in general above, is a combination of two modules. The first one is an off-the-

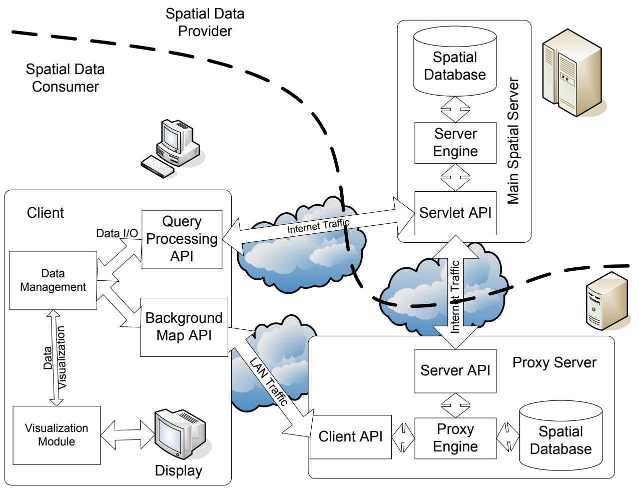


Figure 1.3: Dynamic Proxy — The proxy server is installed with no data on it initially. It connects to the central spatial server and if a request comes from a client for data not available locally, the proxy retrieves the data from the central server, caches it locally, and sends it back to the client.

shelf SQL database[3] responsible for storage of spatial data storage used in handling of window queries. Note however, that the SQL database does not have any information regarding what data it contains compared to the content of the main spatial server. This is the responsibility of the second module, it maintains information about which parts of the SQL database are currently valid (i.e., which parts fully mirror the content of the central SAND database). Additionally, it facilitates communication with the clients and, in case of the dynamic proxy, with the SAND server that performs the role of the central spatial database.

The second module in essence implements a second database which maintains the information about which area of the “world” that is stored in the central database is covered in the local SQL database. The SAND Proxy utilizes the Region Quadtree (e.g., [14, 15, 16]) data structure to manage this information. The problem of determining which areas of the world are represented in the SQL database translates into evaluating window queries on this data structure. The Region Quadtree allows the SAND Proxy to identify quickly and efficiently which part of the main database is available through the local SQL database.

When the proxy server is first started, the auxiliary SQL database is empty and the region quadtree is correspondingly all ’white’. As the clients start

[3] The SAND Internet Browser system has been used with MySQL [4] and PostgreSQL [9] but other SQL databases could also be plugged in.

connecting and requesting spatial data, the proxy server initially forwards these requests to the central spatial data server as it does not store the required information locally yet. Once the data arrives over the network back to the proxy server, the Java code in the application layer fetches the data from the communication layer and inserts it into the database through its JDBC connection. Once the data is stored in the database, it means that the gaps in the coverage are filled. At this point, the local database can be queried directly and the result is then returned back to the respective SAND Internet Browser clients.

For any query window R, some data overlapping the window may already be available locally and some may not be. Therefore, for every window query R, we first test whether the data overlapping R is available locally in full by recursively traversing the region quadtree. If all the data for R is fully available, then no download from the central server is needed. The local database can be used to fetch all the overlapping objects and the resulting data stream can be sent back to the client. If the region quadtree reports that some data overlapping R is missing in the local database, then a download of all the data overlapping R in its entirety is requested from the server. While it will re-load some data that are already present locally, the benefit is that the overhead is much smaller, as only a single window query is submitted to the central server. Any would-be duplicates are ignored by the SQL databases as the data table structure is set up to enforce uniqueness of individual objects stored. This ensures that we do not store duplicate entries in the cache. The decision to aggregate multiple smaller queries into a single larger one is one of the aspects of our design.

After the data overlapping R is loaded from the server, the region quadtree is updated to mark R as fully loaded. This is done through top-to-bottom insertion into the region quadtree — that is, by recursively visiting all overlapping nodes, marking them as covered if they are fully overlapped. Or, in case of a partial overlap and unless the maximum depth was reached, the node is subdivided into four children and the same operation is performed recursively. If there is still just partial overlap of R and leaf node N when the algorithm reaches the maximum allowed decomposition level, then we mark the node as covered. This ensures that any subsequent window query that is simply a result of a lateral movement (i.e., a scroll operation) along the same axis as the window edge that intersects N won't report missing data due to the same N and cause another download request to the central server. Of course, the drawback is that the region tree reports N as available in the SQL database while part of the data overlapping N is in fact missing. In reality, this area is very small (a fraction of the node on the region quadtree maximum depth level) and will typically be loaded before the data is needed — once the window R moves such that it overlaps N in full. This is because N's empty neighbors will trigger download of data overlapping R thus filling the gap in N's coverage as well.

This approach guarantees that the proxy is always able to provide the data requested by the client, while efficiently caching the data for future use. While this approach as described, assumes the auxiliary database has enough resources to store all data that flows through the proxy, it is not a requirement. If the availability of sufficient resources cannot be guaranteed, the same method used in Section 1.3.2 that allows for a limited amount of storage space can be applied here as well.

1.5 Building Combined Solutions

This section describes how the individual building blocks presented previously can be combined together to build a complete spatial database visualization solution. Results of experiments are given that provide guidelines for selection of appropriate designs given specific deployment scenarios.

1.5.1 Modular Design and Chaining

While different host types may be used to cache spatial data, their functionality is similar. Their goal is to store the data that have passed through up to their efficient capacity. The individual proxy modules can be stacked on top of each other, where the node closest to the actual displaying client has the smallest capacity and usually stores a subset of data of its successor in the chain. The farther up in the chain that we go from the client, the more data and processing power the node within the chain has.

This is because when a client requires a certain data range and cannot find this information locally, it sends the request to the next cache/proxy node. If the data is available there, then it is served. If it's not available there, then the cache/proxy requests the same data farther up the chain. This process repeats until the data is reached, in the worst case in the main spatial data server. Once the data is reached, it is sent back the same way the requests came, i.e., all caches/proxies on the way between the client and the successful data repository will get the chance to store the data as well. Since the layers closer to the client have typically smaller capacity, they would usually have to drop some of the data first and thus end up storing subsets of data available on the proxy. This proxy hierarchy is outlined in Figure 1.4. Of course, what data can be expected to be stored on the proxy becomes less clear once the proxy serves multiple clients. In such a case, the proxy may get overwhelmed by requests from another client in such a way that it is forced to drop all data loaded for our client. In this case, our client may still hold some data while the proxy no longer does.

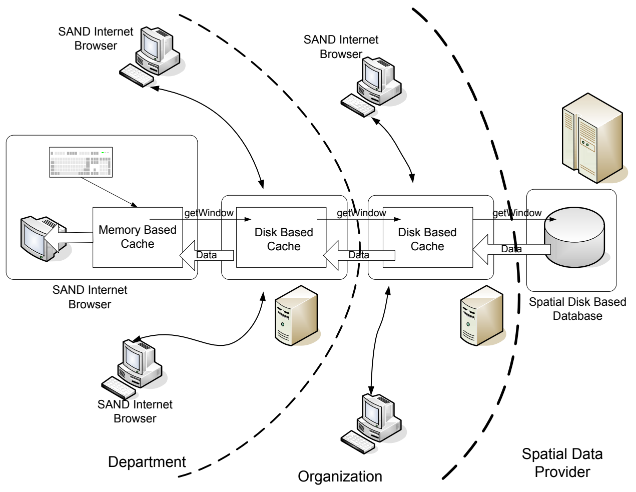


Figure 1.4: SAND Internet Browser and proxies chained together

Regardless of the type of platform managing the data, the data is always stored in a spatial data structure (e.g., some variant of a quadtree). The main data server runs a full-blown spatially-enabled DBMS. The proxies and clients however only perform a subset of operations of a normal DBMS in order to support the limited functionality required by this layered system of caches/proxies.

This layered system is only used for base map visualization. It is not used to evaluate queries. The common interface for nodes participating in the stacked caching system turns out to be very simple:

implements: `getArea(Rectangle area)`
requires: `getArea(Rectangle area)`

This means that each participant in the infrastructure must be able to perform a remote procedure call (RPC) representing a window query on its parent within the hierarchy (where the parent means the node closer to the main server). It also needs to provide a window query interface, i.e., allow nodes closer to the client to submit window queries (RPCs) to it.

Above, we have shown that individual computing platforms can be linked together to create a chain of caching proxies that link the client's visualization module with the central spatial database. Not all computers within this chain need to employ the same caching method. They only need to implement the above interface. The actual implementation can vary depending on the hardware parameters of that platform as well as other factors. However, even within each computer, the individual caching methods do not need to be used in an isolated fashion. The caching concept can be generalized to involve an arbitrary number of caching layers that can be stacked on top of each other in the order of the speed with which they are able to serve the content. Many times, the speed of delivery is inversely proportional to the volume of data any given layer can store efficiently or at all. For instance, access to data stored in primary memory is fast but the storage capacity is limited. On the other hand, a disk-based memory has substantially larger capacity but access to the data is slower.

Note that accessing the central data server can be considered to be within the framework of such a layer as well, and it would be the last and slowest layer; however, it always succeeds (never generates a page fault). So we see that it does not matter whether the data served by any given layer is stored locally or in a remote location. Thus, this concept allows us to generalize the caching into multi-server setups, or even to a peer-to-peer environment. All the client needs to know is in which order it should turn to individual data providing layers. Note that the border between data cache and data server is fuzzy as individual clients can share caches on servers closer to them than the original server, in which case such caches would actually serve as sort of proxies in such environment.

1.6 Evaluation

Our research explores the impact of various types of techniques for chaining different caching layers together on the performance of the solution. We investigated different scenarios and suggest ideal combinations of caching based on the types of devices used, usage model (e.g., number of users looking at the same data), network speed, and other factors.

Specifically, we have designed and implemented the following caching methods and investigated properties of the SAND system created by chaining them in various combinations:

- Client
  1. direct access — client communicates directly with main spatial server with no local caching
  2. local caching — client caches data in its memory
- Proxy
  1. pre-loaded data — the local SQL database is pre-loaded with all spatial data from the server
  2. dynamically-loaded data — the local SQL database is loaded dynamically based on the requests coming from the clients

The behavior of the whole system depends on a number of factors, many outside our reach (e.g., the network latency, number of concurrent users, or even the exact implementation of the garbage-collection algorithm in the underlying operating system or virtual machine, etc.). This also makes a rigorous comparison with other existing systems that aim to serve the same goal (e.g., MapQuest) difficult as we are not able to run performance tests of both systems in identical environments. Therefore, the nature of the SAND system and a MapQuest-type system makes their comparison difficult. Of course, we have tried to minimize the impact of external factors. This is achieved by utilizing the same hardware and software platforms for both systems, the same networking environment as well as identical data sets, queries or sequences of queries. In addition, the parameters of the server platform, the networking environment, and the type of datasets and queries that were run on them were chosen to be typical for the types of deployments that we suggest would benefit from this system.

The goal of this evaluation is not necessarily to determine that one approach is better in every scenario. Instead, we aim to identify what approach is the best one for different methods of deployments and provide the system administrator and user with guidelines for selecting a solution best suitable for their specific needs. Besides comparing vector-based SAND Internet Browser against a bitmap solution, we also deployed SAND in several different ways utilizing its modularity as described in Section 1.5.1.

1.6.1 Comparison with Raster-Based Visualization

For our performance evaluation, we used TIGER datasets from the U.S. Census, specifically the street maps for states in the Mid-Atlantic region. This includes all the roads and streets in Virginia, Maryland, District of Columbia, New Jersey, and Pennsylvania. There are over 7,500,000 entries in this combined dataset. Each entry corresponds to a single line segment, where each actual street may be represented by one or more line segments in the map. The total size of the data stored in the format distributed by U.S. Census is over 700MB.

Our performance testing aims to compare different methods of deploying SAND's vector-based approach to remote mapping with the bitmap based approach employed by such popular systems such as MapQuest. In order to run both systems in the same environment, we chose MapServer [2] to represent the bitmap approach. This allows us to deploy both systems on the same hardware, using the same operating system and within the same networking environment. This also enabled us to minimize performance differences caused by factors that are not directly related to the design of spatial data management.

1.6.2 Typical Usage Scenarios

A user of a mapping or GIS system frequently performs the following operations while navigating the map:

- Zoom in — view an area of interest in more detail.
- Fast Scroll — move the viewable area to the left and to the right, or up and down by large increments. In our scenario, the map moves by one half of the window size, i.e., there is 50% overlap between the old and new views.
- Fine Scroll — move the viewable area to the left and to the right or up and down by small increments, perhaps only by a fraction of the window width or height. In our scenario, the map moves by 10% of the window size, i.e., there is 90% overlap between the old and new views.
- Zoom out — view a larger area of the map within the viewable window.

We expect (and confirm our expectations by running experiments) that the cost of each visualization operation (zoom, pan) for the MapServer approach will be approximately constant given a constant data density (i.e., the number of objects to be visualized for a fixed view area size) and viewable area size. If the number of elements within the viewable area remains the same, then the cost of the spatial query and the cost of subsequent rendering and bitmap transfer remains the same as well. Thus we see that when the number of objects visible as a result of a visualization operation remains the same, the cost of updating remains constant as well. Given a server platform, the MapServer system responsiveness will depend on the network speed and latency. The situation for the SAND Internet Browser is different. There, the system takes a more complex approach when processing visualization requests and the response time depends on the nature of the request as well as on the history of similar requests preceding this one.

As mentioned above, we have selected several typical operations that users of a mapping system or GIS would perform most often while navigating around the map. These operations include zooming in and out and panning/scrolling. First, we compare MapServer with the standard SAND Internet Browser setup that only involves the central data server and the SAND Internet Browser client. Later we also compare MapServer with a deployment of the SAND Internet Browser in an environment where data cannot be stored locally. We conclude with a comparison of the estimated performance of the tile method as typified by Google Maps in the same environment in which both MapServer and the SAND Internet Browser were deployed

For the SAND Internet Browser, we measure the execution time in two scenarios:

- The data to be visualized as a result of the user's operation is already cached on the system.

- The data to be visualized as a result of the user's operation is not yet cached on the system and has to be loaded dynamically from the server.

For MapServer, the bitmap is always downloaded from the server for each new operation.

To measure performance across various deployment scenarios (here represented by different properties of the network connection), we emulate networking environments that correspond to several typical methods of achieving connectivity (i.e., “hookup”) on mobile devices as well as fixed workstations. Table 1.1 describes these connections.

Connectivity (i.e., “hookup”) methods		
Hookup	Bandwidth (kB/sec)	Delay (sec)
Modem	7	0.3
Broadband	182	0.2
Satellite	62	1
LAN	1,250	0.002

Table 1.1: Properties of various network connection (i.e., “hookup”) methods

To emulate standard usage scenarios, all TCP/IP parameters of the networking layer were left at their default values even though for some types of connectivity adjustments of these parameters may improve the overall throughput.

To emulate different networking properties in our test environment, we have utilized NIST Net [6], a general-purpose tool for emulating performance characteristics in IP networks. We have configured NIST Net using networking parameters typical for individual connectivity methods (Table 1.1) to measure the performance of the SAND system in different deployment scenarios.

For the pure client-server environment (i.e., no auxiliary servers), the performance was tested for the following three basic client-server architecture states. First, the *cached* SAND Internet Browser state refers to a scenario where the SAND Internet Browser provides local caching and the data to be displayed as a response to the sequence of scroll operations is already available in the client's memory. Second, the *direct* SAND Internet Browser state refers to a scenario where the client does not cache data locally and downloads all the data from its server. This represents the pure client-server setup where the client communicates directly with the central server. Finally, the *dynamic* SAND Internet Browser state refers to a scenario where the client provides local caching but the necessary data is not available in the local cache yet.

Results of a performance comparison of MapServer with the SAND Internet Browser for fine scrolling can be seen in Table 1.2. During a sequence of fine-scroll operations, the previous window overlaps of the next window 90% of the window area. This means that the SAND Internet Browser can use a fast bitmap copy operation to transfer the part that can be reused to an other location of the screen and it needs to rasterize only 10% of the window using vector data stored either locally or downloaded from the server. We see that the performance in case of cached data is essentially the same across all hookup methods. This is because all data is cached and no data needs to be transferred across the network. Slight differences are due to operations performed by unrelated background processes (e.g., Java VM garbage collection). Also note that while the direct approach does not use any caching and loads all data from from its upstream provider all the time, its data management overhead is lower. Hence, for faster types of network connections the direct method tends to perform better, while the cached methods is typically better for slower connection methods.

Fine scrolling/Local Panning					
Hookup	Cached	Noncached			
	SAND Internet Browser			Map	Est. Tile
	Cached	Direct	Dynamic	Server	Method
Modem	6.6	124	80	179	18
Broadband	6	20	38	52	5
Satellite	5	81	85	181	18
LAN	5	10	33	18	2

Table 1.2: Performance comparison of MapServer, the SAND Internet Browser, and the Tile Method (e.g., Google Maps) for fine scroll. The table indicates the time in seconds to perform 20 subsequent fine-scroll operations.

Results of a performance comparison of MapServer with the SAND Internet Browser for zooming in can be seen in Table 1.3. The starting viewable window showed 25,000 line segments and each zoom-in operation doubled the map scale, i.e., both the x and y coordinate ranges were halved. Thus, the area before the zoom-in operation is four times as large as the area displayed after the zoom-in operation. We measured the time it took to execute five subsequent consecutive zoom-in operations with the last view showing only dozens of line segments.

Note that the viewable area resulting from the zoom-in operation is always a subset of the viewable area that existed prior to the zoom-in operation. Thus, for the caching SAND Internet Browser, the data to be displayed after any zoom-in operation will always be available in the cache. Here we assume that the client uses the same data set on all the zoom levels involved. In practice, zooming in may require the client to display data from additional data layers which may not be available in the cache yet.

Zoom In					
Hookup	Cached	Noncached			
	SAND Internet Browser			Map	Est. Tile
	Cached	Direct	Dynamic	Server	Method
Modem	0.5	10	N/A	44	44
Broadband	0.8	3	N/A	12	12
Satellite	0.5	10	N/A	44	44
LAN	0.8	1	N/A	5	5

Table 1.3: Performance comparison of MapServer, the SAND Internet Browser, and the tile method (e.g., Google Maps) for zoom-in. The table indicates the time in seconds to perform five subsequent zoom-in operations. The results for the dynamic SAND Internet Browser method are not applicable (N/A) since the data will always be cached from the previous operation (assuming all zoom levels retrieve data from the same dataset).

Results of a performance comparison of MapServer and the SAND Internet Browser for the zoom-out operation can be seen in Table 1.4. This query test is essentially a reverse of the zoom-in operation with a single important distinction in the caching SAND Internet Browser. In particular, while each zoom-in operation can expect to have all the necessary data cached from the previous step, in the zoom-out operation this is not necessarily the case. Consider a scenario when the user moves around in a zoomed-in (e.g., street) level and then tries to zoom out (e.g., to city level). As the viewable area grows, not all the data objects that overlap this area are necessarily cached.

For the zoom-out operation, the starting viewable window showed a large detail containing only a few dozens of line segments. Each zoom-out operation expands both the x and y coordinate ranges twice. Thus, the displayed area before the zoom-out operation is four times smaller than the displayed area showing after the zoom-out operation. We measured the time it took to execute five subsequent zoom-out operations, the last view was showing about 25,000 line segments. We considered both scenarios outlined above for the SAND Internet Browser. One scenario captures the situation where the data to be shown after the zoom-out operation is already in the cache (i.e., the zoom-out operation was preceded by a zoom-in operation without any panning operations in between). The other scenario explores a situation when the data to be shown after the zoom-out operation is not in the cache and has to be fetched from the spatial server.

Table 1.5 shows the results of a performance comparison between MapServer and the SAND Internet Browser for global panning. Unlike in the Local Panning/Fine Scrolling scenario evaluated above, in the global panning operation,

Zoom Out					
Hookup	Cached	Noncached			
	SAND Internet Browser			Map	Est. Tile
	Cached	Direct	Dynamic	Server	Method
Modem	1.8	26	48	45	45
Broadband	1.6	5	22	12	12
Satellite	3.2	17	36	45	45
LAN	2.3	2	20	5	5

Table 1.4: Performance comparison of MapServer, the SAND Internet Browser and the tile method (e.g., Google Maps) for zoom out. The table indicates the time in seconds it took to perform five subsequent zoom-out operations.

a large portion of the post-panning viewable area does not overlap the pre-panning viewable area. This means that the SAND Internet Browser must load a large portion of the new viewable area from the locally cached data or from the central spatial server. Given this realization, we again measure the performance of the SAND Internet Browser for two distinct scenarios:

- The data to be visualized as a result of the user's operation is already cached on the system.
- The data to be visualized as a result of the user's operation is not yet cached on the system and has to be loaded dynamically from the server.

MapServer, as always, generates a new bitmap on the server and pushes it onto the client. Each view was showing about 25,000 line segments during this panning operation. As we can see, each of the tests performed above repeats the same operation under the same conditions. This provides us with a comparison of each possible operation under given conditions (in terms of network parameters) separately. While in a real life deployment the network parameters will likely remain fixed during each session, the sequence of operations will probably be a combination of the available operations. In other words, the user will probably not use solely the fine-scroll or the zoom operations, instead they would typically do some scrolling, then zoom in, scroll some more, zoom out, etc. The typical sequence structure and duration of such a session depends on the nature of the scenario. Reviewing a larger area for certain properties may involve much scrolling and a minimum of zooming. Investigation of multiple separate locations may involve more zooming in and out with a minimum amount of panning.

The user will rarely work under conditions when the spatial data is either fully cached all the time or not cached at all in any step. Depending on the exact usage patterns, the user can expect to benefit from the caching for some portion of his or her operations. The success rate of the caching mechanism will depend on numerous factors. The first is the time at which the operation is executed. The cache will be empty right after the start-up

Fast Scrolling/Global Panning					
Hookup	Cached	Noncached			
	SAND Internet Browser			Map	Est. Tile
	Cached	Direct	Dynamic	Server	Method
Modem	3.9	108	109	161	80
Broadband	3.9	19	54	44	22
Satellite	3.9	80	104	165	82
LAN	3.8	9	48	14	7

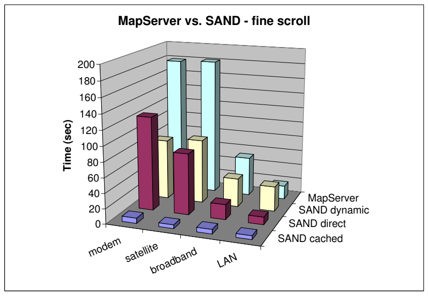
Table 1.5: Performance comparison of MapServer, the SAND Internet Browser, and the tile method (e.g., Google Maps) for fast scroll (global panning). The table indicates the time in seconds to perform 20 subsequent fast-scroll operations.

of the client application. So the user can expect to be fetching data from the server for most such operations initially. Thus, the initial performance of the caching SAND Internet Browser will appear close to what we have shown above under the non-cached data columns (i.e., direct or dynamic). Once the cache is filled with data, the success rate will depend on the extent to which the user's spatial operations are localized. If the user visualizes information directly within the same limited area (e.g., fine scroll or zoom in), then most of the operations will use the cached data. In such a scenario, the performance will be close to what we have shown above in the cached data column. Most of the time the sequence of operations generated by the user will trigger a mixture of cached and non-cached data retrievals. Thus, we can consider our cached and non-cached results as the extreme cases of what a user may expect and a typical experience lies somewhere in between.

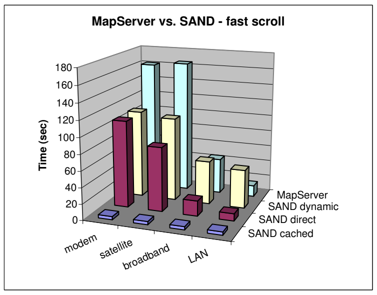
Figures 1.5a–1.5d display Tables 1.2–1.5 graphically. The figures show that in most deployment scenarios, network environments, and usage patterns, the user can expect to have a substantially better experience when using the SAND Internet Browser than when using a pure bitmap system.

1.6.3 Performance Comparisons for Deployments Utilizing Auxiliary Servers

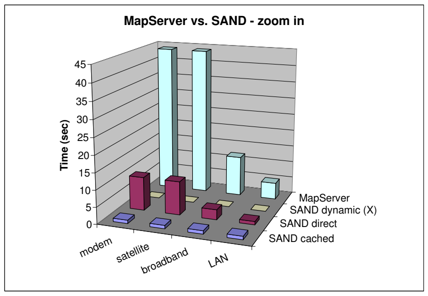
In the previous section we compared the SAND Internet Browser-based system that involved a caching and non-caching client and a central spatial server with a bitmap based system represented by MapServer. Here, we evaluate a scenario outlined in Section 1.4 where a small footprint wireless-capable handheld devices (e.g., smart/cell phones, PDAs and other similar devices) not capable of storing data locally can be used within the SAND Internet Browser-based architecture. Note that the bitmap approach is still valid as



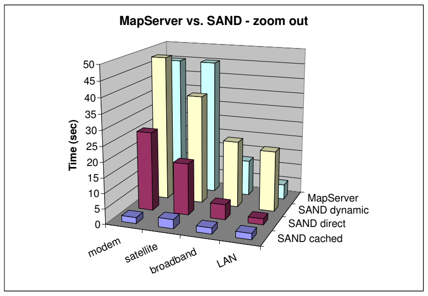
(a)



(b)



(c)



(d)

Figure 1.5: Comparison of a bitmap (MapServer) approach with the vector-based SAND approach for remote spatial data visualization. Assuming all zoom levels retrieve data from the same dataset, note that the SAND dynamic scenario for the zoom-in operation is not applicable as all data is already cached from the previous view (denoted by 'X').

the client does not store any data locally and thus this method is still applicable even on these mobile devices.

We examine three different deployment scenarios. The first scenario involves the static proxy (section 1.4.1) and the state is termed *preloaded*. The remaining two scenarios involve the dynamic proxy (section 1.4.2). The first of the two assumes that the user just started the application so that no cached data is available yet. We call this state *clean*. Since the proxy server can provide its services to multiple users, we also assume that this user is the first one to request this particular data. The second dynamic proxy scenario assumes that the same data was already accessed before (by this or another user), and thus it is already available on the proxy server. This state is termed *cached*.

Using the auxiliary server deployment example from section 1.4 where first responders use handheld devices to communicate with the central facilities via a mobile van, we see that the communication link consists of two parts.

The first link connects the devices and the mobile van, while the second link connects the van with the central facilities. We presume that in emergency scenarios such as these, the connectivity between the handheld devices and the van is faster than the connectivity between the van and the central facility. Based on the assumptions for such an emergency response deployment, we assume that the mobile teams will be able to connect to the central facilities over a satellite link. Locally, the connection between the individual response team members will be wireless (e.g., WLAN 802.11b/g). This emulation is again facilitated by using the NIST Net tool.

Auxiliary Server-based Deployment				
Operation	MapServer	Proxy SAND Internet Browser		
		Preloaded	Dynamic	
			clean	cached
Fast Scroll	165	19	568	52
Fine Scroll	181	29	520	75
Zoom In	44	2	N/A	12
Zoom Out	45	6	58	48

Table 1.6: Performance comparison of various operations for MapServer and the SAND Internet Browser using the auxiliary server deployment method. The scroll operation values represent the time (in seconds) it took the system to process 20 subsequent scroll operations. The values associated with the zoom operations indicate the number of seconds it took the system to process five consecutive zoom operations. The result for the zoom-in operation in the dynamic SAND Internet Browser method is not applicable (N/A) since the data will always be cached from the previous operation (assuming all zoom levels retrieve data from the same dataset).

Table 1.6 shows the results for different usage scenarios that involve auxiliary servers. The client to auxiliary server link is of a wireless LAN type. The link between the auxiliary server and the central spatial server is a satellite connection. Figure 1.6 shows the performance of a system that utilizes auxiliary servers. As we see, MapServer performs better when compared to the SAND Internet Browser on a freshly installed system where no data has been pushed through the infrastructure yet (labeled “clean” in the figure) and the proxy server cache is still empty. This is because of the additional overhead of copying the necessary data from the central server, a step that MapServer completely bypasses. Once the cache is loaded with data, we see that the SAND Internet Browser performs at least as well as, and, most of the time, significantly better than MapServer. If the auxiliary server is preloaded with the data, then the improvement in the performance of the SAND Internet Browser over MapServer is even more pronounced. Note that for the zoom-out operation, the preloaded approach is substantially faster than the cached approach. While there is no network traffic in either case, the cached method has extra overhead (e.g., before sending the data to the client it first needs to verify if any additional data needs to be downloaded from the central server).

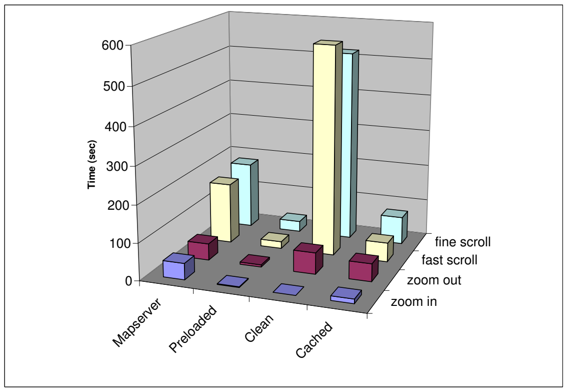


Figure 1.6: Performance comparison of various operations for MapServer and the SAND Internet Browser using the auxiliary server deployment method. Note that the non-cached (clean) scenario for the zoom-in operation is inapplicable.

1.6.4 Comparison with the Tile Method

The tile method, another bitmap-based method providing an alternative to the MapQuest-type bitmap approach, was described in Section 1.2. While we cannot run formal experiments comparing Google Maps or MS Virtual Earth with the SAND Internet Browser directly, we can estimate what their performance would be within the same environment in which MapServer and the SAND Internet Browser were deployed. Assuming that the cost of generating the tiles on the server is negligible, the determinative factor for the cost is the amount of data sent from the server to the client. Since the tile method allows for tile reuse, only the newly visible areas will trigger further download.

When all the necessary data is fully cached from the previous steps, the response times for the SAND Internet Browser and the tile method are essentially instantaneous. For the tile method, the browser simply needs to redisplay the cached tile images which takes no time. Similarly, the SAND Internet Browser also just needs to render and display the cached vector data, done by retrieval and rasterization, which also only takes a fraction of a second (e.g., Table 1.2 once the execution time is divided by 20, the number of scroll operations, to yield the time per fine-scroll operation). Hence, for scenarios where data is fully locally cached, we consider the performance of the tile method and the SAND Internet Browser to be comparable.

Comparing the tile method with MapServer, we find that as the fast-scroll operation reuses 50% of the visible area, the tile method would be twice as fast as MapServer (see Table 1.5. The fine-scroll operation reuses 90% of the area, so the tile applications would be about ten times faster than MapServer (see Table 1.2). Both tables indicate the time in seconds it took to perform 20 subsequent scroll operations. Using the tile method, zoom in/out (if offering the view for the first time and thus not using cached data) would take as long as MapServer as the specific bitmaps are not yet available on the client and thus they have to be loaded from the server in full—that is, they must be reused. Therefore, MapServer results for zoom in/out shown in Tables 1.3 and 1.4 also apply as estimated results of the tile method.

Comparing the tile method with the SAND Internet Browser we find that for fine scrolling (see Table 1.2) the tile method would be faster than the SAND Internet Browser when the data is not already cached (i.e., direct and dynamic). For fast scrolling (see Table 1.5), the tile method would still be slightly faster or perform comparably when the data is not already cached (i.e., direct). The dynamic case is slower in the SAND Internet Browser due to overhead in setting up the caching such as traversing the PMR quadtree, etc. The rationale for the comparable behavior for fast scrolling is that, overall, there is a fixed per-update overhead involved in requesting, handling and storing the data which is higher for the SAND Internet Browser due to the vector format of the data. This overhead is amortized over larger downloads for fast scrolling, thereby making the two methods comparable, while this is not so for fine scrolling, where the overhead makes the performance of the SAND Internet Browser worse than the tile method.

For zoom in/out, the SAND Internet Browser would be considerably faster than the tile method (Tables 1.3 and 1.4). However, in typical deployments, different zoom levels would be displayed with different levels of detail. For instance, when performing a sequence of zoom-in operations, we may need to load more data after some of the steps, even when using the SAND Internet Browser. The advantage of the SAND Internet Browser is maximized when users work with the same level of detail while zooming in and out.

Overall, we expect the tile-based approach to perform similarly to the SAND Internet Browser in many actual usage scenarios. The tile method's drawback is that all of the work is concentrated on the server so as the number of clients connecting to a server rises, performance decreases more rapidly than for the SAND Internet Browser where the client is responsible for more work. Also, the tile method does not allow for development of more sophisticated clients that would execute more operations locally. While for the SAND Internet Browser, the client stores the vector data and can thus perform many operations (such as window or nearest neighbor operations);

for the tile method, the client only has access to the bitmap tiles, which do not provide data for such localized operations. So, we see that the SAND Internet Browser is a better platform for developing smarter, more independent client applications.

1.7 Conclusions and Future Research

We presented a new vector-based system for remote access to spatial databases that could be used where the traditional raster-based approaches do not work too well. We compared the performance of a bitmap raster-based system with the vector-based SAND Internet Browser system. Our experiments allow us to suggest the best type of a remote spatial data visualization tool for a given deployment scenario.

We have developed a modular design for the infrastructure that facilitates remote spatial data access. We applied it to realize several specific types of the SAND Internet Browser system deployment. The best-performing deployment depends on the environment in which the system is to be used. Generally, the system can either be deployed so that the clients communicate directly with the central spatial server. Alternatively, in situations where the client runs on a thin platform or where the service is shared among several co-located clients, an auxiliary server could be used to improve the overall solution's performance.

Future research directions include investigating methods for caching frequently used data in the form of bitmap tiles instead of vectors. While these tiles would only be usable in given views (in terms of zoom factor and layers displayed), they would also allow skipping of repeated rasterization steps. The result would be a hybrid between the SAND Internet Browser and the tile method used by Google Maps and Microsoft Virtual Earth.

References

1. Google Maps. `http://maps.google.com`.
2. MapServer — open source development environment for constructing spatially enabled internet-web applications. `http://mapserver.gis.umn.edu`.
3. MSN Virtual Earth. `http://virtualearth.msn.com`.
4. MySQL — the world's most popular open source database. `http://www.mysql.com`.
5. NASA World Wind. `http://worldwind.arc.nasa.gov`.
6. NIST Net — National Institute of Standards and Technology network emulation package. `http://snad.ncsl.nist.gov/itg/nistnet`.
7. TopoZone — the web's topographic map. `http://www.topozone.com`.
8. MapQuest: Consumer-focused interactive mapping site on the web. `http://www.mapquest.com`, 2002.
9. PostgreSQL — the world's most advanced open source database, 2004. `http://www.postgresql.org/about/`.

10. T. Barclay, J. Gray, and D. Slutz. Microsoft TerraServer: a spatial data warehouse. In *Proceedings of the ACM SIGMOD Conference*, W. Chen, J. Naughton, and P. A. Bernstein, eds., pages 307–318, Dallas, May 2000.
11. A. Harwood and E. Tanin. Hashing spatial content over peer-to-peer networks. In *Australian Telecommunications, Networks and Applications Conference*, pages 1–5, Melbourne, Australia, December 2003.
12. Y. Luo, X. Wang, and Z. Xu. Component-based WebGIS and its spatial cache framework. In *Lecture Notes in Computer Science 3129*, pages 186–196. Springer-Verlag, Berlin, Germany, January 2004.
13. R. C. Nelson and H. Samet. A population analysis for hierarchical data structures. In *Proceedings of the ACM SIGMOD Conference*, San Francisco, May 1987.
14. H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, Reading, MA, 1990.
15. H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.
16. H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan-Kaufmann, San Francisco, CA, 2005.
17. H. Samet, H. Alborzi, F. Brabec, C. Esperança, 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.
18. E. Stolte, C. von Praun, G. Alonso, and T. Gross. Scientific data repositories: designing for a moving target. In *Proceedings of the ACM SIGMOD Conference*, pages 349–360, San Diego, CA, June 2003.
19. C. Yap, K. Been, and Z. Du. Responsive thinwire visualization: Application to large geographic datasets. In *Proc. 14th Ann. Symp., Electronic Imaging 2002*. IS&T/SPIE, 2002. 19-25 Jan, 2002, San Jose, California.
20. R. Zimmermann, W.-S. Ku, and W.-C. Chu. Efficient query routing in distributed spatial databases. In *Proceedings of the 12th ACM International Workshop on Advances in Geographic Information Systems*, I. F. Cruz and D. Pfoser, eds., pages 176–183, Washington, DC, November 2004.