# Shepherdable Indexes and Persistent Search Services for Mobile Users

Michael Higgins, Dominic Widdows, Magesh Balasubramanya, Peter Lucas, David Holstius
{higgins,widdows,balasubramanya,lucas,holstius}@maya.com

MAYA Design Inc.**

*8th International Symposium on Distributed Objects and Applications (DOA), Montpellier, France, Oct 30 - Nov 1, 2006*[1]

**Abstract.** We describe a range of designs for supporting rich search queries in a peer-to-peer network. Our implementation is based upon uniquely identified data objects which are replicated upon request by agents called Shepherds. Several abstract data structures are built upon this framework, supporting dataset management, lexical search, and distributed GIS interfaces in an application called the Geobrowser. Our results demonstrate that it is possible to layer higher-level data structures upon a basic peer-to-peer transport and replication layer. When users perform a given query, parts of the index as well as the query results themselves are shepherded to the user's local venue. A natural benefit of this approach is that mobile users can repeat previous searches if they become disconnected from the rest of the network. Some of the data structures that prove to be successful are peer-to-peer adaptations of traditional indexing structures. We review some of the properties that lead to successful designs in this domain, giving examples of deployed systems in the Geobrowser.

## 1 Introduction and Related Work

This paper presents methods by which indexes supporting rich content-based search interfaces can be built in a peer-to-peer system, by layering sophisticated index structures on a common object location and routing infrastructure.

Cooperative systems for storage and collaboration have experienced a surge of investment and activity over recent years. Of particular interest are flexible decentralized systems, such as Gnutella [1] and OceanStore [2], where nodes may join and leave at any time, and where peer-to-peer replication ensures the high availability of data. One of the major drawbacks to these systems, however, is the lack of cooperative indexing and querying techniques. Users expect to be able to search these systems just as they search the Web, with the same net

** This work was supported by Defense Advanced Research Projects Agency grant SB031-008 for the Cluster-Based Repositories and Analysis project.

performance and quality of results, even though current Web search engines are supported by coordinated and/or centralized systems built on guarantees unmet by peer-to-peer storage. The general information retrieval methodologies outlined for coordinated distributed systems, on which much research has been conducted [3, 4], are not directly applicable.

In peer-to-peer systems, basic one-dimensional queries such as keyword and prefix search are currently supported, but in much the same manner as routing or messaging (see, for example, [5–8]), with logarithmic guarantees of low order. Nodes answer queries about data objects that they store, and route queries they cannot directly answer to a more knowledgeable host. Hybrid techniques improve performance by balancing local work with global work, e.g., by sharing a global keyword lexicon.

Replication of search structures, such as B-trees, is not new [9–11]. Direct storage of search structures in a persistent object store, where chunks of data have unique identifiers but no type, has also been examined [12]. Experiments with replicated search trees have shown that they lead to high availability, overcoming some of the bottlenecks associated with hierarchical structures [13, 10].

But even in systems that make use of index replication, the indexes are generally treated in a way that is distinct from other data — index components are not freely replicated in the same epidemic manner,[2] partly because the information they carry is tied to specific nodes in the system. One approach that *does* propagate index $(key, value)$ pairs epidemically in a peer-to-peer network is the Passive Distributed Indexing system of Lindemann and Waldhorst [11, 14]. While this work is the nearest in spirit to our approach, it still treats index replication and document propagation separately using different infrastructure.

We take a different approach. Instead of having support for a few specific query types (such as keywords or prefixes) built into the infrastructure, we propose a deliberate layering of search functionality on top of the basic object location and routing infrastructure. Our system models index structures using exactly the same attribute-value syntax as is used to model other data objects, such as those used to represent objects in the physical world. Optimistic replication [15] is used to to disseminate the components of a shared index, just like any other data.

Our experiments demonstrate the potential of this approach in an application called the Geobrowser, which is a portal to a peer-to-peer "Universal Database" network [16]. As in the early Visage system [17] and its descendants, all information objects in this network are represented in *u-forms*, a u-form being a mutable bundle of attribute-value pairs identified by a universally unique identifier or UUID. In the peer-to-peer version of this architecture, u-forms are optimistically replicated upon demand to many different venues in the network by artificial agents called *Shepherds*. An index that enables persistent search operations (in the sense that in performing a search, a user will have the data

[2] In this context, the term *epidemic* is used to describe replication strategies in which a new version of an object can be transmitted between any two venues that are in contact.

necessary to repeat that search replicated to their local venue) is described as *Shepherdable*.

This approach carries immediate benefits. Since replication is a core component of peer-to-peer systems, our system automatically gains the advantages afforded by current peer-to-peer storage layers, such as high availability [18] and local caching to support offline activity. Since u-forms are extensible and have dynamic schemata [19], it is always possible to introduce new data structures enabling specialized query functionality or optimizations in performance. In this paper we describe specific implementations of B-trees [20] and R-trees [21], scalable collections (that can be used to support many structures including inverted indexes [4]), as well as a novel self-indexing spatial structure that supports the Geobrowser's distributed mapping application. Each of these structures has been built and optimized using standard u-form data objects, without requiring any modification to the underlying storage and transportation layers.

We also believe that separating content-driven indexes from the storage and transportation layers will have long term benefits. This is because our system will be able to take advantage of improvements in peer-to-peer storage and transportation, and advances in these layers will automatically improve indexing and search reliability and performance with minimal or no changes to application layer code.

The long term success of the Geobrowser platform depends on collaborative addition of data, which implies collaborative contributions to datasets and indexes. Prior work with replicated structures assumes either the existence of a coordination mechanism for contributors to a shared index: a load balancer [10, 9]; a locking mechanism; or a recovery mechanism [22] to ensure that actions on the global index can be effectively serialized. Our general approach to collaboration with u-forms is discussed in more detail in [23].

The rest of this paper is structured as follows. Section 2 describes the underlying database architecture used in our implementations. Section 3 begins to describe higher level u-form data structures, with a design for distributing large collections of u-forms into several navigable chunks. Section 4 describes the use of shepherdable B-trees for search in a totally ordered keyspace, and Section 5 describes a shepherdable R-tree structure with optimizations for distributed spatial search. Section 6 gives the results of some preliminary tests of the R-tree data structure against a large reference collection. Section 7 describes the novel self-indexing structure used for the Geobrowser's distributed mapping environment. Section 8 gives a brief summary of some of the issues involved in creating and maintaining our distributed index structures, and Section 9 analyzes some of the data structures that were prototyped but performed poorly in the Shepherds system.

## 2 U-forms, Replication and Shepherding

This section describes the underlying database architecture used in our applications, emphasizing some of its most relevant properties. All of the implementa-

tions described in this paper use a "Universal Database" architecture in which all information is represented in "u-forms", a u-form being an extensible bundle of attribute-value pairs identified by a universally unique identifier or UUID. A more detailed view of u-forms and the VIA repository is given in [16].

A formal similarity has been suggested between the (UUID, attribute, value) triple and the (URI, predicate, value) structure of the Semantic Web's RDF graphs [24] (and numerous other "triple store" structures for knowledge representation). It is important to note that a u-form in itself is an abstract data object, and physical copies of u-forms can be serialized using many formats, including (for example) XML. For efficiency reasons, our repositories represent u-forms using a recursive bytecode serialization called VSMF (Visage Standard Message Format).

The u-form architecture developed out of the Visage projects for collaborative information visualization [17, 25], and the policy of representing information in u-forms is referred to as the Visage Information Architecture (VIA). The architecture is currently used to support commercial information collaboration applications such as CoMotion and CPOF (Command Post of the Future, a command and control system used by the US military [26]), as well as MAYA Design's Geobrowser and a variety of community information websites [27]. An important requirement of the Geobrowser design is that any information a user accessed in the past when online should still be available offline, along with the search functionality used to locate and select that information.

An important property of u-forms is that their values may contain relations (that function as pointers or external references) to other u-forms. U-forms themselves are by definition location independent, and thus may be replicated to many different venues in the system. This includes the local venue of a user whose contact with the network is through a mobile and sporadically connected device. Over the years, our implementations have increased in scale, making the adoption of peer-to-peer implementations naturally appealing. The adaptation of the u-form concept to a peer-to-peer environment is managed by artificial agents called *Shepherds*. Client applications interact with the Universal Database by reading attributes of u-forms from their local repository interface, and writing the values of certain attributes. The shepherds system makes this possible by performing the following actions:

- If a user or client requests a u-form which is not present in that user's local repository, the shepherds try to locate a copy in the network and replicate an instance of that u-form to the user's local venue.
- The shepherds try to ensure that all replicas of a u-form hold the same contents.

Instances of u-forms may be located in the network by a variety of indexing techniques such as manually configured routing tables, centralized Google-like indexing services, and/or distributed hash-tables [28, 29]. Considerable research and development has been carried out by ourselves and others on this problem, and while it is difficult, initial solutions exist and it is reasonable to believe that

they are not bounded in scale. For the purpose of this paper, we assume that the storage layer works efficiently and scales effectively.

Nonetheless, some operations are costly in our system. These include:

- **Following "null" references.** If a client somehow generates or receives a UUID of a u-form that does not exist anywhere in the network, the process of requesting this u-form causes expensive and unrewarding network traffic, also causing high latency for the user before a "null" answer is returned.
- **Conflicts.** If two users update the same u-form concurrently, this will generate a conflict when the shepherds realize that there are two incomparable versions of the the u-form.

In general, it is difficult to resolve u-form conflicts automatically, and a more stable approach is to design information architectures for collaboration that distribute authority and avoid conflicts on the same u-form [23]. This is of relevance when trying to organize writes to shepherdable indexes.

## 3 Managing Collections and Datasets

In this section, we begin to give examples of distributed data structures created out of u-forms. In the earliest examples of Visage, shared collections are represented in a single collection u-form containing several UUID references in its `members` attribute. Users can add/remove u-forms to/from a collection using drag and drop. The whole collection can be dragged onto a map frame or a bar chart, enabling a user to see and select objects in certain geographically or statistically defined regions.

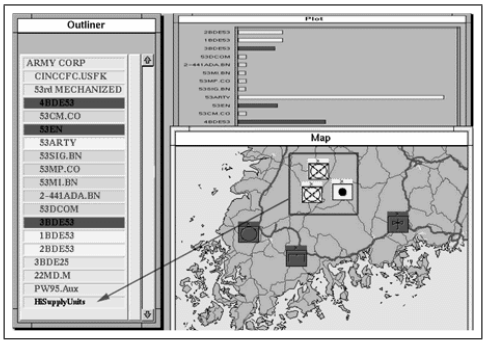


**Fig. 1.** Visage users morph collections of objects from one display to another

Unfortunately, the method scales poorly. This is because the basic operation of reading or writing an array of UUIDs to the repository is linear in the length of the array. For large collections, a better method is to break such collections up and distribute their members over several linked u-forms. One simple way to do this is to chunk the collection into smaller collections of (e.g.) 100 members each, and create a linked list structure out of the new chunks. By adding extra information to this data structure, performance can be enhanced considerably, as follows:

- A pointer from the head to the tail of the collection can guarantee (amortized) constant time append.

– By recording an associative hash of the member UUIDs on the head node, the fact that edits have been made can be recorded, and checked by other users without scanning the collection.
– By adding a B-tree structure [20, p. 262] or skip list pointers [30], access to any part of the collection can be provided in $O(\log N)$ space and time.
– Given suitable annotation and refactoring of child node counts or skip probabilities, an entire collection can be added recursively to another collection.

The combination of these techniques leads to the creation of a u-form data structure that we call a "Navigable Recursive Scalable Collection". Such a collection, displayed in the Geobrowser interface that enables peer-to-peer browsing, navigation and editing of the collection, is shown in Figure 2.

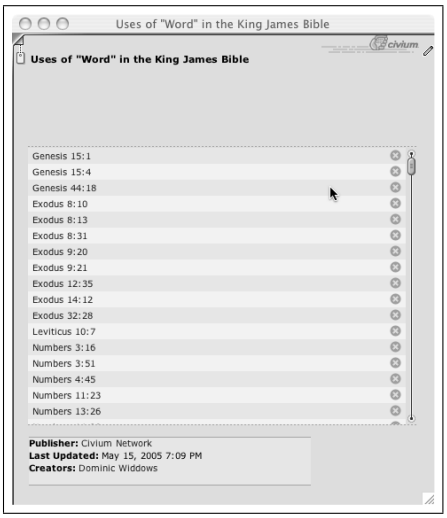


**Fig. 2.** Interface enabling distributed edits of a collection

In large collections, the positions for items in scalable collections are computed from knowledge of how many descendants each individual node in the B-tree has, and so B-tree nodes in scalable collection implementations have to carry enough information about how many descendants they have for quick comparison with their siblings.

Such a design is typical of the issues involved in adapting complex data structures to a peer-to-peer environment for distributed collaboration. A scalable design for building and editing a collection can be achieved using traditional techniques such as B-trees [20, pp. 262-265] or skip lists [30]. Neither method requires all of the member u-forms to be present in any one location, and enables the construction of user interfaces that perform well for the collection management operations. The choice between these data structures depends on intended usage patterns. We have found the B-tree representation to be more space efficient for static collections. On the other hand, skip lists (being a probabilistic data structure) are more robust in the case where several users in different venues may be editing the collection frequently and asynchronously.

The Geobrowser interface demonstrates that the scaling problem can be solved so that a collaborator in a peer-to-peer network can have apparent access to a dataset containing several millions of u-forms, while only needing to shepherd the u-forms they are interested in and a handful of navigational u-forms that enable the user to find information effectively.

## 4 Distributed B-tree Indexes for Lexical Search

Once a user has access to a large distributed dataset, it is natural to wish to find (for example) all u-forms in a collection whose `name` attribute contains a particular word. Such functionality is provided using the support of a B-tree structure [20, pp. 262-265], represented in u-forms to make a shepherdable index.

The main difference between our use of B-trees for building search trees, and their use for managing navigable scalable collections in the previous section, is that the records in search trees are key-value pairs, whereas in the scalable collection, the records are simply values (UUIDs of member u-forms) whose 'key' is effectively its position in the collection. The search tree behaves like a distributed associative array or dictionary, whereas the scalable collection behaves like a distributed array.

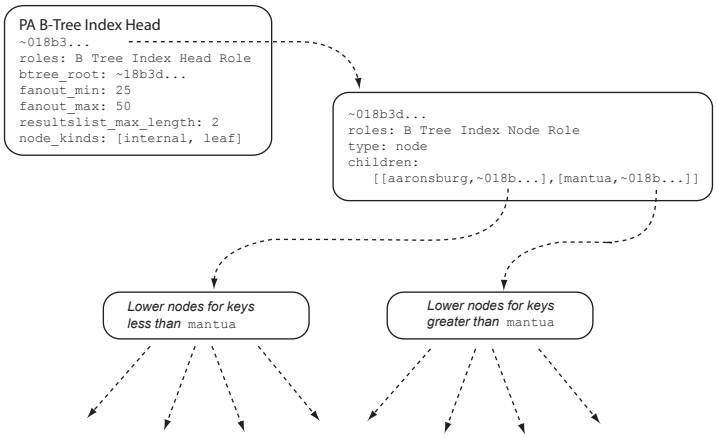


**Fig. 3.** Upper nodes in B-tree of names of populated places in Pennsylvania

An example u-form B-tree for searching for populated places in Pennsylvania according to their `name` attribute is shown in Figure 3. To search the tree for a particular key such as *pittsburgh*, one proceeds from the head node to the root node, at which point the keys *aaronsburg* and *mantua* are available. Since *pittsburgh* is between *mantua* and the end of the alphabet, the next u-form requested is the one whose UUID is listed after the *mantua* key in the root node, and so on.

The most important attribute that enables this descent is the `children` attribute, which stores a sorted array of (*key*, *UUID*) pairs. The rule is simply that the query should follow the UUID relation of the highest key that precedes the query expression.

The search starts at a separate "head node" rather than a constant "root node", because B-tree root nodes are occasionally split and subsumed due to

rebalancing. The head node records structural data about the tree, including the maximum and minimum fanout allowed in nodes below the root.

There is also a `resultslist_max_length` attribute that is used for deciding how many results to pack into a leaf node, based upon how many values match a particular key. For example, the query term *highland* matches with the name attribute of 19 places in the dataset, and since this is a reasonably large number, this collection of results is posted in a separate u-form. On the other hand, the term *versailles* only occurs in 2 placenames, and as an optimization, these 2 results are placed directly in the leaf node.

This is typical of a standard trade-off in information retrieval, between the structure of the key-space or lexicon, and the structure of inverted indexes or 'postings lists' (see e.g., [4, §4]). In choosing to push some of the postings into the lexicon itself, we are reducing latency for infrequent lexical items, at little extra up front space cost.

Other applications of the B-tree data structure for lexical searching operations are as follows.

- **Full text search.** Utilizes the B-tree data structure to keep a lexicon, and the scalable collection of Section 3 to store the inverted indexes or postings lists. Extra information for more sophisticated indexing, including term-weights and offsets, can be placed in attributes parallel to the members of the postings list collection.
- **Boolean semantics for collections.** The scalable collection design of Section 3 does not enable an efficient Boolean membership test. Such a requirement can be supported by creating and maintaining an auxiliary B-tree whose keys are the UUIDs of the collection, ordered lexically.
- **Virtual references.** Another use of UUIDs as keys is to record when one u-form has referred to another. The use of indexes of this sort to store virtual references in support of backpointers, standoff annotation and publisher-contributed content is described in [31, 23].

## 5 Distributed R-tree Indexes for Spatial Search

This section describes the design and use of a shepherdable index for low-dimensional continuous data, which enables the Geobrowser to dynamically find and display local content from large geospatial and geotemporal datasets.

The most stable solution to this problem has in practice been a distributed R-tree approach. R-trees were introduced by [21] and are described along with related structures in [32, §6.3]. In an R-tree, the index keys are bounding boxes in an $N$-dimensional key space, and are ordered by containment. That is, the children of a particular tree node $A$ are index nodes or data items whose bounding boxes are contained in the bounding box of $A$. New items can be added to the index by adding them to the node whose bounding box volume is minimally increased by the insert operation, and nodes are (heuristically) split by trying to minimize overlap and minimize the amount of "dead space", that is, the volume within each bounding box that is not actually contained in any of its children.

For many of the datasets we wish to visualize in the Geobrowser, there are some items that are considered more important than others, and these items should be presented first. For example, for a dataset of world populated places, large cities such as *Shanghai* should be presented before smaller towns and villages. In many user interfaces, it is appropriate to limit the number of items shown to (for example) 100 objects from each dataset.

### 5.1 Priority Item Annotation

The user experience can be greatly enhanced by adding prioritized items to higher nodes on the R-tree.[3] Prioritization enables a search to return the $p$ most important items within a particular bounding box, without searching all the way down to leaf nodes in the tree to obtain these items. A simplified example is shown in Table 1, which contains the main attributes and values of the current root node of the u-form index of the world's populated places. This shows the bounding boxes of the children, and the first five prioritized items (*Shanghai, Mumbai, Seoul, Moscow, Manila*). A query that only wants to find the world's five or fewer most populated places only needs to look at this u-form.

**Table 1.** Current root node of World Cities populated places index. The items in italic type are UUID relations to other u-forms.

| | |
|---|---|
| `aggregate_child_count` | 2462019 |
| `bbox` | [[-54.93, -179.98], [90.0, 180.0]] |
| `children` | [[*(no name)*, [[-51.8, -179.98],[90.0, 179.75]]], [*(no name)*, [[-54.93,-160.20], [70.96, 178.87]]], [*(no name)*, [[-47.05, 66.95], [36.4833333, 180.0]]]] |
| `priority_items` | [[*Shanghai*, [[31.22, 121.45],[31.22,121.45]], 0.992], [*Mumbai*, [[18.97,72.82], [18.97, 72.82]],0.991], [*Seoul*, [[37.56, 127.00],[37.56, 127.00]], 0.990], [*Moscow*, [[55.75, 37.61], [55.75,37.61]], 0.990], [*Manila*, [[14.60,120.98], [14.60,120.98]], 0.989] |

Another simple way to optimize query performance is to annotate child pointers with an extra property listing the maximum importance or priority score of any item that can be reached via that node. This is called *Maximum Importance Annotation*, and is used to prune searches when the Geobrowser interface specifies that it will only display information above a certain priority or importance score.

[3] Note that adding priority items to higher nodes is different from the "prioritized R-tree" data structure of [33], which optimizes worst-case search by adding prioritized *nodes*, rather than optimizing simple cases by adding prioritized *items*.

## 6 Reference Dataset and Performance Tests

The development work described in this paper was carried out by MAYA Design as part of the Information Commons and Geobrowser projects. The immediate purpose of this effort was to create an index with good query performance to enable Geobrowser users to access to find places from the Information Commons Gazetteer [34], a public resource of populated places and worldwide administrative divisions, containing fused information about over 5 million populated places and their political affiliations. The bounding boxes in the R-tree index for this dataset are shown in Figure 4, which gives an interesting statistical picture of world population distribution.

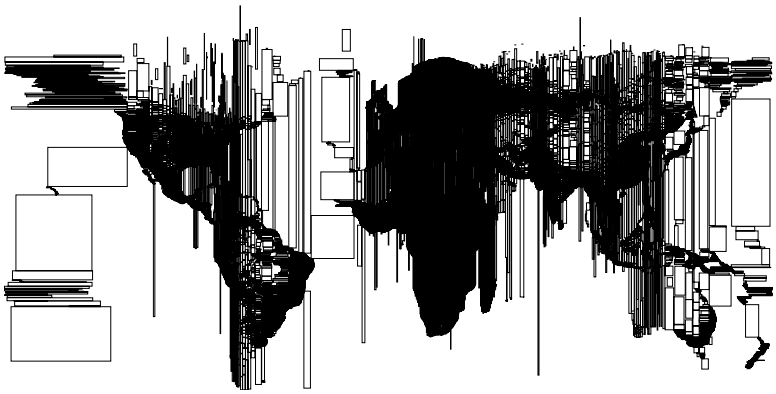

**Fig. 4.** Leaf-nodes in the R-tree representing World Populated Places.

Because the path through the index is replicated as well as the results of the query, subsequent queries can make use of relevant portions of the index. Table 2 shows how some simple queries are affected by this caching property. For each test condition, we measured the time cost of performing a query on an empty local venue, thus inducing index replication over the network. We then wiped the local venue of all u-forms, and performed a related query. Without wiping the local venue, we performed the original query again, and measured the speed-up gained by being able to take advantage of cached index structure from the related query. Replication was performed over a wide-area network. The test query was a geo-spatial bounding box. The related query types were:

**Repeated** The related query was identical to the test query, so all of the index structure could be re-used.

**Sub** The related query was a bounding box properly containing the test query. Again, all of the index structure could be re-used.

**Expanded** The related query was a bounding box properly contained in the test query. Only some of the index structure could be re-used.

**Adjacent** The related query was a bounding box adjacent to the test query. Only some of the index structure could be re-used.

**Distant** The related query was a bounding box very distant from the test query. Very little of the index structure could be re-used.

**Table 2.** Results summary of R-tree query performance experiments

| Related Query Type | Initial Query(s) | Subsequent Query(s) | Speed-Up Factor |
|---|---|---|---|
| Repeated | 4.5 | 0.77 | 5.8x |
| Sub | 2.8 | 0.53 | 5.4x |
| Expanded | 3.6 | 2.1 | 1.8x |
| Adjacent | 4.1 | 1.7 | 2.4x |
| Distant | 7.7 | 8.2 | 0.95x |

When we can re-use a significant portion of the index structure, performance is naturally much better. When we can re-use less, a corresponding drop in performance improvement is seen. When we cannot re-use any significant portion of the index, we see that natural variance in the testing overwhelms any benefit from caching: the subsequent query for the "distant" condition is actually slightly worse than the original.

Given space constraints, we elide measurements of similar performance improvements for B-tree based structures. The measured performance improvements are more noticeable in real applications for geo-spatial queries. We believe this is because that query locality is more natural in the geo-spatial domain than it is for, e.g., keyword searches.

## 7 Distributed GIS Data and Recursive Self-Indexing

The most novel index structure we have so far built upon the u-forms and shepherds infrastructure supports distributed mapping in the Geobrowser. Not only point data items, but the vector shapes of the base maps themselves, are represented in u-forms and found by following references to new u-forms and requesting these u-forms through the shepherds system.

Naturally, a list of vector points can easily be expressed as an attribute of a u-form, so geometric objects can be represented in u-forms. The design challenge in this domain is to do this in such a way that shapes can be suitably factored and composed to present a persistent and appealing user interface in a peer-to-peer network. Basic requirements of the Geobrowser map interface include:

1. It should be possible to drag out a spatial object into its own frame, add it to a collection, and add comments, just as with other phenomena in Visage interfaces.
2. The user should see a basic outline map of the area of interest as soon as possible.
3. More detailed shape data should be rendered as the shape u-forms become available.

4. The user should be able to zoom in to obtain greater levels of detail in specific areas.

Requirement 1 is supported by representing the shape of each object in a Cartesian coordinate frame that is optimized for the object in question. (For example, with shapes on the earth's surface such as boundaries of landmasses and political subdivisions, the approximate centroid is located, and then the local east, local north, and outward normal are used as orthonormal $x-$, $y-$ and $z$-axes.)

Requirements 2, 3 and 4 are met in the following way. A simplified outline shape is calculated using the Douglas-Peucker line simplification algorithm [35]. This outline shape then stores relations to more detailed line segments, along with information saying which points in the main shape should be replaced by the extra detail in the subshapes. An example of this approach to shape rendering is shown in Figure 5, which depicts the Supercontinent (Asian, African and European landmasses) with both the first level (roughest) and second level (slightly more detailed) decomposition. The first level points are contained in a single u-form, whereas the second level points are broken across 10 different u-forms of roughly similar length. In Figure 5, both levels are depicted together for explanatory purposes, though in the actual Geobrowser, the second level data actually replaces the first so that there is a single coastline. The benefit is that the user only needs to shepherd in the u-forms containing more detailed information for the parts of the world that are of interest. Once accessed, these local detail u-forms stay available the user for as long as they are wanted.

To support all of this functionality, shapes must be able to include one another recursively. Sometimes shapes are included without replacement (for example, adding an island to a continent in requirement 1), and sometimes subshapes replace parts of their parent shapes (meeting requirement 3). In both cases, a parent shape creates an annotated collection of child shapes, for each child shape giving:

1. The bounding box of the child shape in the parent's coordinate frame; and
2. The linear transformation used to map the child points into the parent frame.

In this way, shape u-forms act as containers for lists of geometric points, and as an index to subsidiary shapes. These subsidiary shapes may be topologically distinct, or may be more detailed parts of the main shape itself. In this way, a shape u-form acts as an index to its own parts, and is described as a *self-indexing* structure.

A rendering algorithm proceeds by:

1. Testing the bounding boxes in turn to see whether the shape intersects with the user's field of view; and
2. Retrieving the child points, and using the linear transformation given to map the points to the parent frame, from which they are mapped to screen coordinates.

**Fig. 5.** Shape of the Supercontinent (orthogonal projection), showing first and second levels of detail combined.

These mappings are all calculated using standard linear algebra as used in many computer graphics algorithms (see e.g., [36, §5.6]), and can be optimized significantly by hand-coding the floating point operations for our specific use case. Since all shapes use a form of Maximum Importance Annotation (as described in Section 5.1), we can control the amount of information presented to the user, and a judicious choice of relative priorities and important features enables the Geobrowser to present a high quality user experience with acceptable latency in the interface.

The results are demonstrated in Figure 6, a screenshot from a version of the Geobrowser delivered to the Greater New Orleans Non-Profit Knowledge Works (`www.gnonkw.org`) as part of the New Orleans redevelopment effort following the Hurricane Katrina disaster. This interface actually combines many of the technologies described in this paper. The category tree on the left hand side is modelled using a scalable collection: so by using the interface shown in Figure 2, users can easily add new categories to the map and new items to a category. Each of these scalable collections also has a u-form R-tree index, so the items that intersect with the user's map panel can be efficiently identified. Shapes such as the coastline, the water features, and the roads, are all represented using the recursive u-form shape format described in this section: once an object is returned by the R-tree index, if it has a geographic shape attached, the extended shape of the object can be rendered.

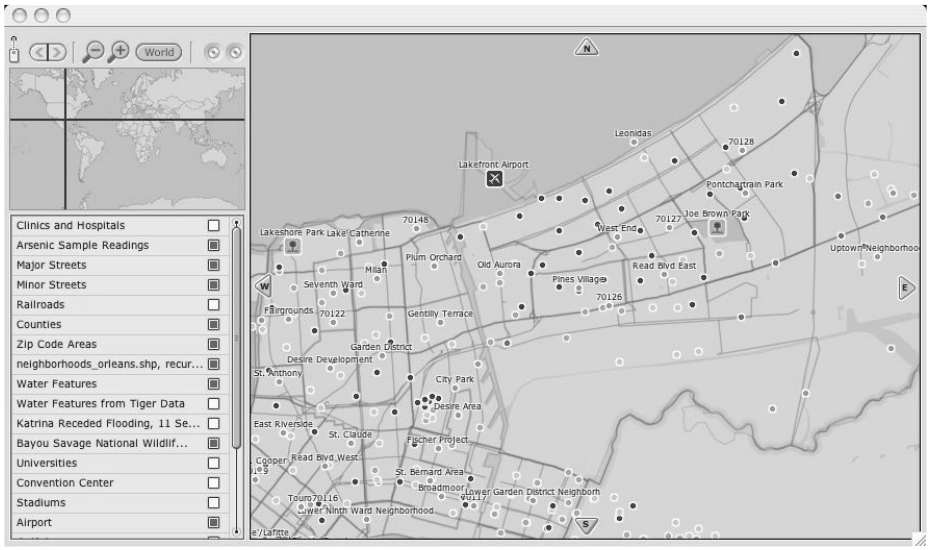


**Fig. 6.** Map showing part of New Orleans (orthogonal projection), with data such as locations of streets, parks, and levels of arsenic concentration.

An interesting side effect of our approach to shape simplification and reassembly is that it can reduce the data required and hence the computational complexity of performing a point in polygon test, which is traditionally an $O(n)$ operation, or at best, $O(\log(n))$ for convex polygons [37], where $n$ is the number of vertices in the polygon. Recommended optimizations for point in polygon tests involve (for example) dividing the line segments into quadrants and finding integer versions of the necessary arithmetic [38].

The Douglas-Peucker simplification produces a "tolerance" measure $\varepsilon$ along with the simplified shape, where $\varepsilon$ is the maximum distance that a point in the original shape is from the simplified shape. Using standard $O(n)$ methods, it is easy to write an algorithm that (i.) tests whether a test point is inside or outside the simplified polygon, and (ii.) measures the minimum distance from the test point to the simplified polygon. If this distance is greater than $\varepsilon$, then the result also holds for the fully detailed polygon. Otherwise, the u-forms containing more detailed points for the nearby segments are requested and the test is repeated. While this algorithm produces appreciably faster results, formal analysis of this algorithm's complexity is complicated, because it depends to a large extent on the choice of appropriate levels of simplification, which itself depends on number of points, point density, average curvature, and the performance / detail tradeoff.

The more general issue is that logical agents should behave in robust ways given naturally limited resources. Finite resources include computational power and time, and also the data required to solve a problem [39]. It is especially important with distributed information systems to design solutions that can

answer a user's questions with the smallest possible bandwidth requirements and the maximum flexibility in terms of *where* computation is performed.

## 8 Multiple Writers and Keyspace Locality

The shepherdable designs presented in this paper have focused on enabling efficient distributed read access to indexes. It is also necessary to enable many writers to contribute information, which leads to concerns about concurrency and consistency. Managing decentralized contribution and collaboration in u-forms is a large topic that is discussed more fully in [23]. Due to space constraints. this section presents only summary results that are relevant to the index structures we have discussed.

The simplest way to have many users contributing to a shared index is to have them all write to a common data structure with a single root node. Unfortunately, u-forms in this data structure soon become conflicted, especially if any of the users are offline. This is compounded by the fact that offline users usually only have access to parts of the index that they have read from, and so lack the ability to read another part of the keyspace to decide where in the index to put a new item. UUIDs that are manufactured by concatenating an agreed prefix with a semantically meaningful suffix can help users to put new data in predictable u-forms, but this only compounds the problem that having multiple writers to the same u-form generates conflicts. The generation of conflicts is not an insurmountable barrier, but does lead to some difficult design considerations (see [23]).

One way to solve this problem is to have index writes managed by agents in specific, reliable venues. Requests for items to be added are made by using u-forms and shepherds to mediate an asynchronous messaging system [23, §4.1]. Users should keep their proposed additions in a separate index, which can be flushed when a query to the main shared index demonstrates that their addition has been successful. This in part reduces the problem of managing decentralized index changes to the problem of managing large scale centralized indexes, to which there are known solutions [4]. To distribute load for large indexes, it is sometimes possible to partition the keyspace into different regions covered by different indexing services without greatly affecting the balance of the tree as a whole.

In some cases, the approach of partitioning the index into different parts of the keyspace works particularly well, because there is close correspondence between some parts of the keyspace and the activity of certain user groups. This is particularly likely in geospatial indexes, where users are often focused on a particular region of interest, that corresponds closely to a particular branch or branches of the index. (Compare this with an alphabetic B-tree, where it is unlikely that a user maintains interest in words beginning with a certain string of letters over a long period of use.) Keyspace locality also enhances querying both online and offline, which we believe explains the relative reuse benefit of R-tree searches (Section 6). To gain similar bulk reuse of u-forms for lexical search,

topical datasets have to be carefully designed, which can reduce either the scope or the performance of general system-wide searches.

## 9 Rejected Designs

This section gives a brief summary of designs that we prototyped but eventually rejected because they performed poorly or had undesirable features.

- **Single U-Forms.** Making a single large "index" u-form whose attributes correspond to the terms in the index performs poorly for large vocabularies.
- **UUID concatenation.** One can create a "base UUID" $U$ for the index as a whole, and find the index entry for a particular term $T$ by manufacturing the UUID $U+T$. This performs well for positive searches, but very badly for out of vocabulary items, because of the 'manufactured null UUIDs' problem (see section 2).
- **Quad tree.** A spatial index structure that used manufactured UUIDs resulting in similar performance problems was the quad tree (see [32, §6.2]).
- **Trie structure.** Digital search trees or trie structures [20, Ch 17] proved to be reasonably effective in our system, though the dependence of query latency on the length of the query terms lead to relatively poor performance compared with the B-tree. Also, the natural support of B-trees for range queries is a benefit for one-dimensional search of continuous data, which is used by one of our community directory websites (`http://goguide.3rc.org/`) for calendar search. A similar design for spatial search (where more general quad tree bounding boxes kept pointers to those of their children that were occupied) was similarly more effective than the regular quad tree, but poor compared with the more balanced R-tree.

## 10 Conclusion

We have presented a range of shepherdable data structures that support efficient rich-query peer-to-peer search for a variety of interesting applications. We achieve this flexibility by creating search trees and other data structures out of basic database objects, relying on the underlying infrastructure for location, routing and replication. This strategy enables Geobrowser users to repeat previous searches even if they lose network connectivity.

Applications of this technology are not limited to keyword searches, but can be used to create a variety of sophisticated user interfaces such as distributed mapping. At the same time, the careful structuring of information in shepherdable indexes can sometimes encourage developers to construct algorithms that perform more efficiently in a fragile network, by giving results based upon partial data that may nonetheless be sufficient.

Writing to shepherdable indexes is more complex, because naive implementations lead to conflicts, which introduces issues not all of which have yet been solved. To enable reliable decentralized index writes, some combination of service centralization and index partitioning may be necessary.

## References


1. Ripeanu, M.: Peer-to-peer architecture case study: Gnutella network. Technical report, University of Chicago (2001)
2. Kubiatowicz, J., Bindel, D., Chen, Y., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., Zhao, B.: Oceanstore: An architecture for global-scale persistent storage. In: Proceedings of ACM ASPLOS, ACM (2000)
3. de Kretser, O., Moffat, A., Shimmin, T., Zobel, J.: Methodologies for distributed information retrieval. In: International Conference on Distributed Computing Systems. (1998) 66–73
4. Melnik, S., Raghavan, S., Yang, B., Garcia-Molina, H.: Building a distributed full-text index for the web. In: World Wide Web. (2001) 396–406
5. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. Lecture Notes in Computer Science **2218** (2001) 329
6. Stoica, I., Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H.: Chord: A scalable Peer-To-Peer lookup service for internet applications. In: Proceedings of the 2001 ACM SIGCOMM Conference. (2001) 149–160
7. Harvey, N., Jones, M.B., Saroiu, S., Theimer, M., Wolman, A.: Skipnet: A scalable overlay network with practical locality properties. In: In proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS '03), Seattle, WA (2003)
8. Aberer, K., Cudré-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., Schmidt, R., Wu, J.: Advanced peer-to-peer networking: The P-Grid System and its Applications. PIK Journal - Praxis der Informationsverarbeitung und Kommunikation, Special Issue on P2P Systems (2003)
9. Johnson, T., Colbrook, A.: A distributed data-balanced dictionary based on the b-link tree. In: Proceedings of the 6th International Parallel Processing Symposium, Washington, DC, USA, IEEE Computer Society (1992) 319–324
10. Cosway, P.R.: Replication control in distributed B-trees. Technical Report MIT/LCS/TR-705 (1997)
11. Lindemann, C., Waldhorst, O.: A distributed search service for peer-to-peer file sharing in mobile applications. In: Proc. 2nd IEEE Conf. on Peer-to-Peer Computing (P2P 2002). (2002)
12. Crespo, A., Garcia-Molina, H.: Archival storage for digital libraries. In: Third ACM International Conference on Digital Libraries. (1998)
13. Kroll, B., Widmayer, P.: Distributing a search tree among a growing number of processors. In: SIGMOD '94: Proceedings of the 1994 ACM SIGMOD international conference on Management of data, New York, NY, USA, ACM Press (1994) 265–276
14. Lindemann, C., Waldhorst, O.P.: Exploiting epidemic data dissemination for consistent lookup operations in mobile applications. SIGMOBILE Mob. Comput. Commun. Rev. **8** (2004) 44–56
15. Saito, Y., Shapiro, M.: Optimistic replication. ACM Computing Surveys **37** (2005)
16. Lucas, P., Senn, J., Widdows, D.: Distributed knowledge representation using universal identity and replication. Technical Report MAYA-05007, MAYA Design (2005)
17. Roth, S., Lucas, P., Senn, J., Gomberg, C., Burks, M., Stroffolino, P., Kolojejchick, J., Dunmire, C.: Visage: A user interface environment for exploring information. In: Proceedings of Information Visualization, San Francisco, IEEE (1996) 3–12

18. Viles, C.L., French, J.C.: Dissemination of collection wide information in a distributed information retrieval system. In: SIGIR '95: Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval, New York, NY, USA, ACM Press (1995) 12–20
19. Lucas, P., Widdows, D., Hughes, J., Lucas, W.: Roles in the universal database: Data and metadata in a distributed semantic network. Technical Report MAYA-05009, MAYA Design (2005)
20. Sedgewick, R.: Algorithms in C. Addison-Wesley (1990)
21. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: Proceedings of SIGMOD. (1984) 45–47
22. Lomet, D.B., Salzberg, B.: Concurrency and recovery for index trees. VLDB Journal: Very Large Data Bases **6** (1997) 224–240
23. Higgins, M., Roth, S., Senn, J., Lucas, P., Widdows, D.: Managing distributed collaboration in a peer-to-peer network. 14th International Conference on Cooperative Information Systems (CoopIS 2006) (2006)
24. Manola, F., Miller, E.: RDF primer (2004)
25. Higgins, M., Lucas, P., Senn, J.: VisageWeb: Visualizing WWW Data in Visage. In: Symposium on Information Visualization (Infovis), IEEE (1999) 100–107
26. Project, D.: Command post of the future (CPOF) (2005) `http://www.darpa.mil/ato/programs/CPOF/DT.htm`.
27. Allegheny County Department of Human Services: HumanServices.net (2006) `http://www.humanservices.net/`.
28. Stoica, I., Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H.: Chord: A scalable Peer-To-Peer lookup service for internet applications. In: Proceedings of the 2001 ACM SIGCOMM Conference. (2001) 149–160
29. Li, J., Stribling, J., Morris, R., Kaashoek, M.F.: Bandwidth-efficient management of DHT routing tables. In: Proceedings of the 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI '05), Boston, Massachusetts (2005)
30. Pugh, W.: Skip lists: A probabilistic alternative to balanced trees. In: Workshop on Algorithms and Data Structures. (1989) 437–449
31. Balasubramanya, M., Higgins, M., Lucas, P., Senn, J., Widdows, D.: Collaborative annotation that lasts forever: Using peer-to-peer technology for disseminating corpora and language resources. In: Fifth International Conference on Language Resources and Evaluation (LREC 2006), Genoa, Italy (2006)
32. Rigaux, P., Scholl, M., Voisard, A.: Spatial Databases. Morgan Kauffmann / Academic Press (2002)
33. Arge, L., de Berg, M., Haverkort, H., Yi, K.: The priority r-tree: A practically efficient and worst-case optimal r-tree. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD '04), Paris, France (2004) 347–358
34. Lucas, P., Balasubramanya, M., Widdows, D., Higgins, M.: The Information Commons Gazetteer: A public resource of populated places and worldwide administrative divisions. In: Fifth International Conference on Language Resources and Evaluation (LREC 2006), Genoa, Italy (2006)
35. Douglas, D.H., Peucker, T.K.: Algorithms for the reduction of the number of points required to represent a line or its caricature. The Canadian Cartographer **10** (1973) 112–122
36. Foley, J.D., van Dam, A., Feiner, S.K., Hughes, J.F.: Computer Graphics. Addison Wesley (1990)

37. Haines, E.: Point in polygon strategies. In Heckbert, P., ed.: Graphics Gems. Volume IV. Academic Press (1994) 24–46
38. Hormann, K., Agathos, A.: The point in polygon problem for arbitrary polygons. Computational Geometry **20** (2001) 131–144
39. Gabbay, D.M., Woods, J.: The New Logic. Journal of the Interest Group in Pure and Applied Logics **9 (2)** (2001) 157–190