Spatial Search in Networked Systems

Misbah Uddin, Rolf Stadler
ACCESS Linnaeus Center
KTH Royal Institute of Technology
Email: {ahmmud,stadler}@kth.se

Alexander Clemm
Cisco Systems
San Jose, California, USA
Email: alex@cisco.com

***Abstract*—Information in networked systems often has spatial properties: routers, sensors, or virtual machines have coordinates in a geographical or virtual space, for instance. In this paper, we propose a peer-to-peer design for a spatial search system that processes queries, such as range or nearest-neighbor queries, on spatial information cached on nodes inside a networked system. Key to our design is a protocol that creates a distributed index of object locations and adapts to object and node churn. The index builds upon the concept of the minimum bounding rectangle, to efficiently encode a large set of locations. We present a search protocol, which is based on an echo protocol and performs query routing. Simulations show the efficiency of the protocol in pruning the search space, thereby reducing the protocol overhead. For many queries, the protocol efficiency increases with the network size and approaches that of an optimal protocol for large systems. The protocol overhead depends on the network topology and is lower if neighboring nodes are spatially close. As a key difference to works in spatial databases, our design is bottom-up, which makes query routing network-aware and thus efficient in networked systems.**

***Index Terms*—network search, spatial search, distributed spatial index, distributed query processing.**

I. Introduction

In our recent work, we introduced the concept of network search, which builds upon an information-centric view of network management and is specifically suited for large-scale, dynamic environments [1]. A network search system allows operational and configuration data in a networked system to be searched in real-time using keywords and relational operators. We developed and implemented a peer-to-peer design of such a system, using a self-organizing network overlay without centralized components. Search queries can be initiated from any node in the overlay. Data that can be searched is organized as objects that are maintained in real-time databases inside search nodes close to the data source. Search queries are propagated across nodes to the target objects. Network search is also suitable for environments where bandwidth constraints or privacy/compliance restrictions prevent the export of data to an external system.

Often, information associated with networked systems has spatial semantics. There are many use cases that involve searches for nearby objects, for instance. Examples include applications that search for servers within a maximum network delay range or a surveillance system that searches for the k sensors that are closest to a given geographic location.

This motivates us to introduce spatial concepts in network search and study spatial queries for search in networked systems. In our work, we use the well-known Euclidean spatial model in $\mathbb{R}^n$, which defines locations as n-dimensional vectors, spatial entities, such as lines, hyper-rectangles, n-spheres, etc., and a distance function. Once we associate locations with objects, we can formulate spatial queries.

First, consider a networking scenario where locations for routers, servers, and virtual machines are produced by a network coordinate system, such as Vivaldi [2]. In this case, the Euclidean distance between locations refers to the approximate round-trip time between the network entities at those locations. Spatial queries include finding a server that is closest to a client application or finding a server with a similar distance to a given set of clients. Second, consider a networking environment where locations refer to geographic coordinates and distances refer to geographical distances. A spatial query for this case is finding backup servers outside a given area. Third, consider an IoT scenario with garbage containers at different geographical locations. A spatial query is finding full containers within a certain distance from a given place. Fourth, consider an ICT infrastructure, whereby locations are IP addresses mapped onto $\mathbb{R}^4$ (in case of IPv4). A spatial query in this case is finding physical or virtual machines in a given address range, for the purpose of security management. Lastly, consider the case where a router searches for the closest gateway in terms of delay or other QoS parameters in the context of performance-based routing.

The objective of our work is to design a decentralized spatial search system for a Euclidean spatial model, which defines the locations of objects and the distance between spatial entities. The system should support a range of spatial queries. The design should include a distributed query processing architecture and a protocol that routes queries to local databases that store the objects inside the network. Query processing should be efficient and scalable. Our approach is as follows: first, query processing is realized through a peer-to-peer system of search nodes which store the objects; second, an object-location index that is distributed over the search nodes allows for efficient query processing. While the overall goal of our research is to extend the network search paradigm with spatial concepts, this paper focuses exclusively on designing and evaluating a spatial search system and leaves out integration aspects.

With this paper, we make the following contributions. We propose a peer-to-peer design for a spatial search system. The design includes a protocol that creates and maintains a distributed index of object locations, which is based on the

978-3-901882-77-7

R-tree concept [3]. We present a generic query processing protocol, which is based on an echo protocol and uses the index to perform pruning of the search space and query routing. We give an example of a specific spatial query that can be processed using this protocol. Our design is adaptive in the sense that the distributed index is updated in response to changes to local databases and to changes in network topology. By design, the response times of spatial queries increase with the diameter of the network, which is asymptotically optimal. Extensive simulations show the effectiveness of the index in reducing the overhead of the query processing protocol, when compared to a baseline protocol that searches all nodes for a given query.

All spatial search systems we found in the literature follow a top-down design [4], [5], [6], [7]. In these systems, the index information, which is the basis for query routing, is created top-down, by recursively partitioning the location space. The index is then used to create an overlay topology that optimizes query routing in terms of query execution time and/or query overhead on the overlay, without taking into account the underlying network. In addition, indexes contain only pointers to information objects, and how these objects are retrieved and processed is not addressed in these works. The spatial search system presented in this paper is unique in that it follows a bottom-up design. First, the index information is created bottom-up, from a large number of local databases on interconnected search nodes. Second, network links between these nodes directly relate to the underlying routing or physical topology, in order to enable efficient query routing in the target environment. Third, a characteristic of our design is that the objects are stored on the same node as their indexes, and object information can be retrieved locally as part of query processing.

II. Spatial Queries

We present two well-known classes of spatial queries that often appear in use cases.

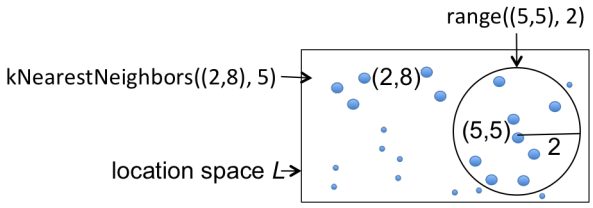


Fig. 1: Spatial queries on a location space $L \in \mathbb{R}^2$. Locations appear as dots. The query range((5,5), 2) returns all objects with locations in the disc with center (5,5) and radius 2, the point query range((5,5), 0) returns objects at (5,5), and the query kNearestNeighbors((2,8),5) returns objects with locations represented by larger dots outside the disc.

We assume a search space O of objects, whereby each object $o \in O$ has a location l in the Euclidean space $L = \mathbb{R}^n$. Two objects o and p in this space have the Euclidean distance $d(o,p) = \sqrt{\sum_{i=1}^{n}(o_i - p_i)^2}$.

We consider two classes of spatial queries on L. The queries in the first class, called *range* queries, return all objects in O whose locations intersect with a given geometric object like a point, a line, a hyper-rectangle, or an n-sphere. For example, the query range(l,r) returns all objects within a distance $r \geq 0$ from a location l. A range query with $r = 0$ is also called a *point* query. The query outsideRange(l,r) returns the complement of range(l,r) in $\mathbb{R}^n$. The second class of queries we consider are the *nearest-neighbor* queries, which return the k objects in O whose locations have the shortest distance from a given geometric object. The query kNearestNeighbors(l,k), for example, returns the k objects nearest to a given location l. Searching for k objects nearest to the centroid of a given set of locations is another example of a nearest-neighbor query. Figure 1 shows examples of spatial queries.

III. Architecture

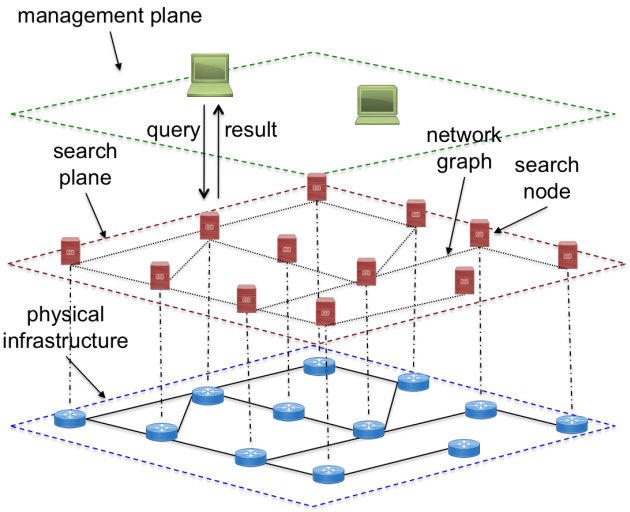
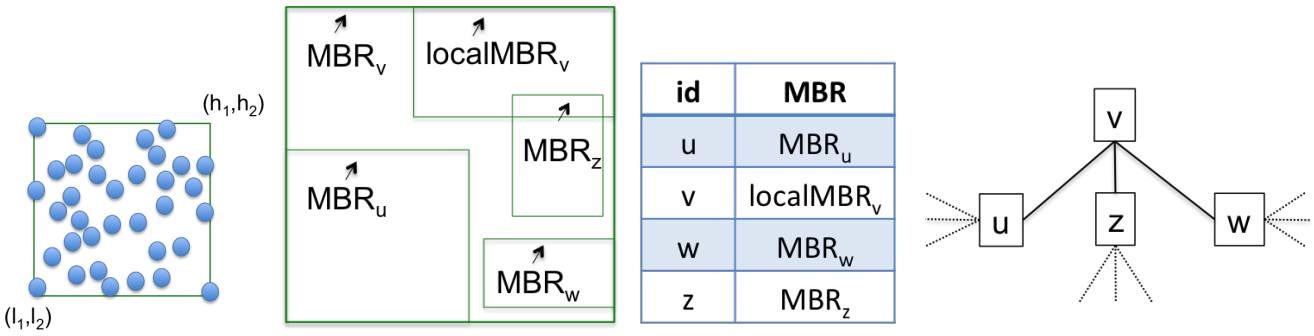


Fig. 2: The architecture of a spatial search system.

Figure 2 shows our architecture for a spatial search system. Its key element is the *search* plane, where spatial queries are processed. This plane contains a bidirectional, connected graph G of *search nodes*. The vertices of the graph are the search nodes and the edges are the message-passing links. Each search node maintains a local database of objects. It performs the processing of spatial queries in collaboration with other nodes. During query processing, it exchanges messages with its neighbors of the network graph. The search plane forms a peer-to-peer query processing system where a query can be invoked at any node and its result is obtained from there.

The bottom plane in Figure 2 represents the physical infrastructure. Each element of this infrastructure is associated with a search node, which maintains information from it in form of objects. The top of the figure shows the management plane, which issues the queries and processes the results.

We envision that a search node runs on the physical infrastructure, for instance, on a network element, on a server, or



(a) (b) (c)

Fig. 3: (a) The MBR of a set of locations in $\mathbb{R}^2$ shown as dots; the locations (l_1, l_2) and (h_1, h_2) defines the MBR, (b) MBR_v is the minimum bounding rectangle of MBR_u, MBR_w, MBR_z, and localMBR_v, (c) The local index of node v (left); node v and its neighbors on the spanning tree (right).

on a network appliance. The links between search nodes are congruent to link-level/routing-topology connections.

IV. Distributed Index

In this section, we propose a distributed index structure of object-locations for processing spatial queries. We also present the protocol *MBRIndex*, which builds and maintains this index. In the following, we use the term node instead of the term search node, as shown in Figure 2.

The index is based on the geometric concept of the *Minimum Bounding Rectangle* (*MBR*). An MBR is a hyper-rectangle in the location space L. It is defined by two locations, $(l_1, \cdots, l_n)$ and $(h_1, \cdots, h_n)$, whereby $l_i \leq h_i$. We use an MBR to aggregate the locations of the objects in a node's local database, as well as the locations of objects a node knows about. Given a set of locations, the MBR that aggregates this location information is the hyper-rectangle with the smallest diameter that includes all of these locations (see Figure 3(a); for better readability, we assume locations in $\mathbb{R}^2$). The MBR concept allows us to efficiently capture a potentially infinite number of object locations by giving just two locations.

MBR is a common concept for indexing spatial information in a variety of areas [8]. It is generally used in form of an R-tree [3]. Each node of such a tree is associated with an MBR, whereby the MBR of a non-leaf node is the minimum bounding rectangle that includes the MBRs of each children. R-trees are used in spatial databases as part of local or distributed indexes [9], [6]. Our design distinguishes itself from all other distributed designs we found in the literature in that, in our design, each node is the root of an R-tree. When describing our design, we use the term *MBR-tree* instead of R-tree, since the latter term is associated with a specific top-down space partitioning method.

In our design, the distributed index is built bottom-up. Each node has a *local MBR*, namely, the MBR of the object locations in the local database. To build an MBR-tree, we first construct a spanning tree on graph G (see Section IV-B), which defines the topology of the MBR-tree. On this spanning tree, *the MBR of a leaf node* is the local MBR, and *the MBR of a non-leaf node* is the minimum bounding rectangle of the MBRs of each of its children, as well as its local MBR. As an example of a non-leaf node, MBR_v in Figure 3(b) is the minimum bounding rectangle of MBR_u, MBR_w, and MBR_z, which are the MBRs of its children, and localMBR_v, which is its local MBR. In our design, an MBR tree forms the basis for query processing and routing for those queries that are invoked at the root node of this tree.

The MBR of a node v of the MBR-tree provides information about the object locations v has knowledge of. Therefore, in our design, MBR-trees form a basis for query processing and routing.

Figure 3(c) shows the local index T of a node v. It is a table with two columns, where each row contains information about a node, either about v or one of its neighbors on the spanning tree. The first column contains the local id of a node; the second column contains the MBR of that node. When node v processes a query, it uses the MBR field of each of its neighbors on the spanning tree to decide whether or not to forward, i.e., route the query to that neighbor.

A. The Indexing Protocol

Algorithm 1 gives the pseudocode of the indexing protocol, namely, MBRIndex. This protocol runs on each node and initializes and updates the MBR fields of the local index. During its execution, nodes exchange update messages with neighbors over the links of the spanning tree. The local data structures of a node v executing the protocol are given on top (lines 1-2). The variable N is initialized with the set of neighbors of v on the spanning tree. The update messages exchanged by the nodes during the protocol execution are of the form (UPDATE, n, MBR_n) (line 3), where n identifies the sending node. Node v starts the protocol by initializing the MBR fields of the local index and sending an update message with its local MBR to each of its neighbors (lines 4-7). We use the symbol $(-\infty, \infty)$ for the MBR that represents the entire location space $\mathbb{R}^n$. v then enters a loop where it processes update messages. After reading an update message from u, v updates the MBR field of row u with the MBR contained in

Algorithm 1 *MBRIndex.*
Pseudocode for node v. Protocol executes on a spanning tree.

data structures:
1: N := set of neighbors of v;
2: T := local index table;
messages:
3: (UPDATE, n, MBR_n)
MBRIndex():
4: T.MBR[v] := local MBR;
5: **for** $n \in N$ **do**
6: T.MBR[v] := $(-\infty, \infty)$;
7: send (UPDATE, v, T.MBR[v]) to n;
8: **while** true **do**
9: read (UPDATE, u, MBR_u);
10: T.MBR[u] := MBR_u;
11: **for** $n \in N$ **do**
12: $MBRsend_n$:= MBR($\{T.\text{MBR}[i]_{i \in N \cup \{v\} - \{n\}}\}$);
13: send (UPDATE, v, $MBRsend_n$) to n;

the message (lines 9-10). After that, for each neighbor n, node v computes the minimum bounding rectangle of all MBRs in the local index, except the MBR of the node n, and sends an update with the computed MBR to node n (lines 11-13).

Critical to Algorithm 1 is line 12, which can be illustrated using the example in Figure 3(c). The MBR that node v sends to node u, for instance, is the minimum bounding rectangle of its local MBR, the MBR of node z, and the MBR of node w. In other words, node v sends its node MBR to node u, assuming u is its parent. This MBR represents object locations v has knowledge of.

To keep the pseudocode readable and small, first, we omit in Algorithm 1 some statements that would prevent the protocol from sending unneeded messages. For instance, before sending an update to a neighbor, a node can check whether the MBR to be sent in the update message is different from the MBR sent in the last message. In case there is no difference, the message can be omitted. Second, if a change in the local MBR occurs, we assume that a message (UPDATE, v, $newLocalMBR_v$) is generated, which can be read by the protocol.

B. Constructing the Spanning Tree

MBRIndex executes on a spanning tree of the network graph G. There are several protocols for spanning tree construction that can be used for this purpose, for instance, the distributed Bellman-Ford algorithm [10]. To handle node churn, we need a protocol that reconstructs the spanning tree upon changes in G. Such a reconstruction is possible as long as G remains connected. GAP is an example of a protocol that reconstructs the spanning tree after churn events [11]. GAP is based on a self-stabilizing tree-construction protocol described in [12]. For the evaluation of our design in Section VI, we use an implementation of GAP to create and maintain the spanning tree.

C. Maintaining the Local Index in Response to Node Churn Events

Churn events we consider include a node joining the search system, a node leaving the system, and a node crashing. We outline an extension to the MBRIndex protocol, so that the extended protocol will adapt the local index of each node after a churn event. We rely on a spanning tree protocol that reconstructs the spanning tree after such events (see Section IV-B).

First, we consider a new node u joining the network graph G. Upon joining, u starts the spanning tree protocol and the MBRIndex protocol. Then, u initializes the local index table with a single row $(u, (-\infty, \infty))$. In the meantime, the spanning tree protocol reconstructs the spanning tree such that u is included. As part of the reconstruction process, the spanning tree protocol creates a message JOIN(u) on each neighbor v of u on the spanning tree. Similarly, it creates a message JOIN(v) on node u. The message JOIN(u) on v is processed by the MBRIndex protocol on node v by adding entry $(u, (-\infty, \infty))$ to its local index table. After that, v sends an update message to node u. The message JOIN(v) on node u is processed by the MBRIndex protocol the same way.

Second, we consider a node u (with its adjacent links) leaving the network graph G. After this event, the spanning tree protocol reconstructs the tree. Then, it generates a message LEAVE(u) on all former neighbors v on the spanning tree. the MBRIndex protocol processes this message by deleting the entry for row u in the local index table of node v. After that, node v sends an update message to its neighbors on the spanning tree. In case removing u from G results in a partitioned graph, the MBRIndex protocol reconstructs the index for each partition of G.

Third, regarding the MBRIndex protocol, a node crash can be modeled as a node leaving the system. This means that, after a node u crashes, its neighbor v receives a message LEAVE(u), and the processing of this message will be the same as described above.

D. Properties of the MBRIndex Protocol

The protocol in Algorithm 1 has the following properties, which are based on properties of trees, tree-based aggregation, and echo protocols [13]. We give them without proof due to lack of space. (P_1): The MBRIndex protocol converges in finite time on a finite spanning tree. (P_2): After convergence, for each node, the minimum bounding rectangle of all entries of its local index is equal to the MBR of all object locations. (P_3): After convergence, each node is the root of an MBR-tree whose leaves are the local MBRs. This tree can be easily extracted from the local indexes. Furthermore, once the algorithm has been extended as outlined in Section IV-C, the properties (P_1), (P_2), and (P_3) hold after a churn event, as long as the graph G stays connected.

V. The Query Processing Protocol

SpatialSearch is a protocol that computes the spatial queries discussed in Section II. It is an echo protocol that executes on

Algorithm 2 *SpatialSearch*.
The SpatialSearch protocol executes on a spanning tree S of the network graph G and processes the spatial query $query$. The query is invoked at node v=root. After termination of the protocol, the query result is at the root node in variable R.

data structures:
1: N := neighbors of v on S;
2: $parent$:= null; ▷ parent of node v on S
3: D := database of local objects;
4: T:= local index table;
5: $R := \emptyset$; ▷ (partial) result of query
6: $prunepars$:= null;
message types:
7: (EXP, $sender$, $querypars$, $prunepars$);
8: (ECHO, $sender$, R_{sender});
protocol *SpatialSearch*()
9: **if** $v \neq$ root **then**
10: receive (EXP, n, $querypars$, $prunepars$);
11: $parent := n$, $N := N$ - $\{n\}$;
12: $query$.localQuery($querypars$, D, R);
13: $query$.computePrunePars(R, $querypars$, $prunepars$);
14: $query$.pruneTree($querypars$, $prunepars$, T, N);
15: **for** each $n \in N$ **do**
16: send (EXP, v, $query$, $prunepars$) to n;
17: **while** $N \neq \emptyset$ **do**
18: receive (ECHO, n, R_n);
19: $query$.updateResult($querypars$, R_n, R);
20: $N := N$ - $\{n\}$;
21: **if** $v \neq$ root **then** send (ECHO, v, R) to $parent$;

the spanning tree introduced in the previous section. It can be started at any node, and this node becomes the root of the protocol execution tree S (which has the sam e topology as the spanning tree). During its execution, the protocol uses the local index to prune the execution tree, in order to reduce the number of nodes whose local databases are searched during the execution of a query (see example in Figure **??**). When the protocol terminates, the result of the query is available at the root node.

Algorithm 2 shows the pseudocode of the SpatialSearch protocol. For better readability, we give the code for the case where the root node is fixed. Further, the code refers to the spatial query in abstract form, whose interface is shown in Algorithm 3. A concrete realization of a spatial query is shown in Algorithms 4.

The code for the SpatialSearch protocol in Algorithm 2 is given for a node v. The local data structures are shown on top (lines 1-6). N is initialized with the set of neighbors of v on S; $parent$ holds the parent of v on S; D is the database of local objects; T is the local index table; R holds (partial) result of the query; $prunepars$ contains the parameters for pruning S.

During the execution of the SpatialSearch protocol, neighboring nodes exchange two types of messages: explorer messages of the form (EXP, $sender$, $querypars$, $prunepars$) and echo messages of the form (ECHO, $sender$, R_{sender}) (lines 7-8). $sender$ identifies the node that sends the message; $querypars$ holds the parameters of the spatial query; $prunepars$ holds the parameters for pruning S; R_{sender} contains the partial result of the query known to $sender$.

The code of SpatialSearch contains both the code for the root and non-root nodes (lines 9-21). The root node starts by performing the method localQuery on database D, which updates the search result R (line 12). The protocol then computes the query-specific parameters for pruning (line 13). After that, it prunes S by removing those neighbors from N that do not contribute to the search result (line 14). Then, v sends an EXP message to all neighbors in the pruned set N (lines 15-16) and waits for an ECHO message from each neighbor in N (line 17). Upon receiving such a message, it updates R by executing the method updateResult and removes n from N (lines 18-20).

A non-root node v, upon receiving an EXP message sets $parent$ to the sender of the message and removes the sender from N (lines 9-11). Then, it performs the operations in lines 12-20 as described above. After that, it sends an ECHO message to $parent$ with the partial result R (line 21).

(For an in-depth discussion of echo algorithms, see [14], [13]).

Algorithm 3 Abstract methods of $query$.
$\downarrow$ refers to an input parameter, $\uparrow$ refers to an output, and $\updownarrow$ refers to a parameter that is both an input and an output.

abstract object $query()$
1: $query$.localQuery($\downarrow querypars$, $\downarrow D$, $\uparrow R$);
2: $query$.updateResult($\downarrow querypars$, $\downarrow R_{child}$, $\updownarrow R$);
3: $query$.pruneTree($\downarrow querypars$, $\downarrow prunepars$, $\downarrow T$, $\updownarrow N$);
4: $query$.computePrunePars($\downarrow R$, $\downarrow querypars$, $\updownarrow prunepars$);

Algorithm 4 *range*(l, r) query.
The query returns objects within distance r from location l.

object *range*(l, r):
1: $l \in \mathbb{R}^2$ location;
2: $r \geq 0$ distance;
3: **procedure** $query$.localQuery($[l, r]$, D, R)
4: R := disc(l, r) $\cap$ D;
5: **procedure** $query$.updateResult($[l, r]$, R_{child}, R)
6: $R := R \cup R_{child}$;
7: **procedure** $query$.pruneTree($[l, r]$, p, T, N)
8: **for** each $n \in N$ **do**
9: **if** T.MBR[n] $\cap$ disc(l, r) = $\emptyset$ **then** $N := N - \{n\}$;
10: **procedure** $query$.computePrunePars(R, $[l, r]$, d)
11: d := null;

Algorithm 3 contains the interfaces of the abstract spatial query. The method localQuery executes a spatial query with parameters $querypars$ on the local database D and returns the

result in R. The method updateResult merges R with R_{child} from a child node. The method pruneTree removes neighbors from N using query-specific parameters and the local index T. The method computePrunePars updates the prune parameters for pruning the tree S.

We give a specific example of a spatial query in the form of concrete implementations of the abstract $query$ in Algorithm 3. The query is range(l, r), which finds objects within distance r from a given location l. Its pseudocode is given in Algorithm 4. The method localQuery updates R with objects in D that are within distance r from the location l. The method updateResult updates R by merging it with R_{child}, which contains the objects known to $child$ that are within distance r from location l. The query range(l, r) does not use specific pruning parameters. The method pruneTree removes the neighbor n from N if the disc defined by parameters r and l does not overlap with the MBR of node n with respect to node v.

VI. Performance Evaluation

We have evaluated the performance of the SpatialSearch protocol using Peersim, a Java-based platform for simulating large-scale peer-to-peer systems [15]. We model a network search system as a (network) graph of search nodes, each having a local database with objects. The SpatialSearch protocol executes on these search nodes and communicates via sending messages over the links of the network graph. To perform the evaluation, we implemented the SpatialSearch protocol for point, range, and kNearestNeighbor queries. We also implemented the MBRIndex protocol, which creates and maintains the distributed location index. Further, we implemented a GAP protocol that creates a spanning tree on the network graph [11]. Finally, we implemented a topology generator that generates network graphs for different network sizes and connectivity policies, as well as a coordinate generator that produces the locations for nodes of a network graph.

Performance Metrics: We evaluate the performance of the SpatialSearch protocol for a specific query by counting the number of nodes whose local databases are searched during the execution of the query, and we compare the result with two baselines. The first baseline is the size of the network graph, which is an upper bound for the performance of SpatialSearch and which is the performance of the protocol that searches all local databases. We call this protocol *CompleteSearch*. It can be implemented as a simple echo algorithm. The second baseline is the minimal number of nodes whose databases must be searched to process a search query. We call such a hypothetical protocol *OptimalSearch*.

In the evaluation, we compute the *efficiency metric*

$$E = \frac{N_{SpatialSearch} - N_{Optimal}}{N - N_{Optimal}}$$

whereby $N_{SpatialSearch}$ is the number of nodes whose local databases are searched by SpatialSearch when processing the query, N is the size of the network graph, i.e., the number of nodes, and $N_{Optimal}$ is the minimal number of nodes to be searched. $E = 0$ is the value for OptimalSearch and $E = 1$ that for CompleteSearch. The metric E allows us to compare the performance of SpatialSearch for different network sizes, and it indicates how well the protocol performs compared to CompleteSearch (upper bound) and OptimalSearch (lower bound). For instance, a value of $E = 0.2$ means that the protocol has an overhead of twenty percents (of the normalized difference between the baselines) relative to OptimalSearch and saves eighty percents relative to CompleteSearch. Further, the metric E not only implies efficiency, but also scalability. A small value of E means that the system has a greater capacity to support a larger number of concurrent spatial queries.

Further, we evaluate the performance of SpatialSearch with respect to execution times of various queries. Due to lack of space, we present details in [16].

Network Graph Topology: For the evaluation, we produce network graphs with 512, 2048, 8192, 32,536, and 131,072 nodes. For each node, we generate a location in $\mathbb{R}^2$ uniformly at random within a square of size s=16 for a network with 512 nodes, s=32 for a network with 2048 nodes, S=64 for a network with 8192 nodes, s=128 for a network with 32536 nodes, and s=256 for a network with 131,072 nodes. Note that the average node density is the same for all network sizes.

We use two different graph topologies in the evaluation. First, the *Barabási-Albert* topology, which is a scale-free topology that follows the degree distribution $P(k) \sim k^{-3}$, $k = 1, 2, 3, \cdots$ [17]. Second, the *nearest-neighbor* topology, which is constructed by connecting each node of the graph to its k nearest neighbors. For the experiments, we set $k = 20$, which produces connected graphs with a high probability. Before the experiments, the graphs are tested for connectivity.

The *Barabási-Albert* topology is oblivious to node locations, while the *nearest-neighbor* topology is constructed based on location awareness. We expect the performance of the protocol on the *nearest-neighbor* topology to be better than on the *Barabási-Albert* topology, since the area of the MBR of a subtree of the spanning tree is in expectation smaller for a *nearest-neighbor* topology than for a *Barabási-Albert* topology. Comparing the performance of the protocol for both topologies with the same number of nodes is fair in the sense that the spanning trees of these topologies have the same number of nodes and edges.

Local MBRs: We assume that each search node is aware of its location and maintains objects in database that are nearby. For simplicity, we assume for the evaluation that the location of all objects in the local database is the same as the location of the node.

A. SpatialSearch for range(l,r) queries

Figure 4 shows the measurement results of the SpatialSearch protocol for range(l,r) queries executing on both topologies and different network sizes. The top two plots in Figure 4 shows measurements on the *Barabási-Albert* topology, the bottom two on the *nearest-neighbor* topology. Figure 4(a) and 4(c) contain efficiency vs distance curves, for different network sizes. $E = 0$ is the efficiency of OptimalSearch and $E = 1$

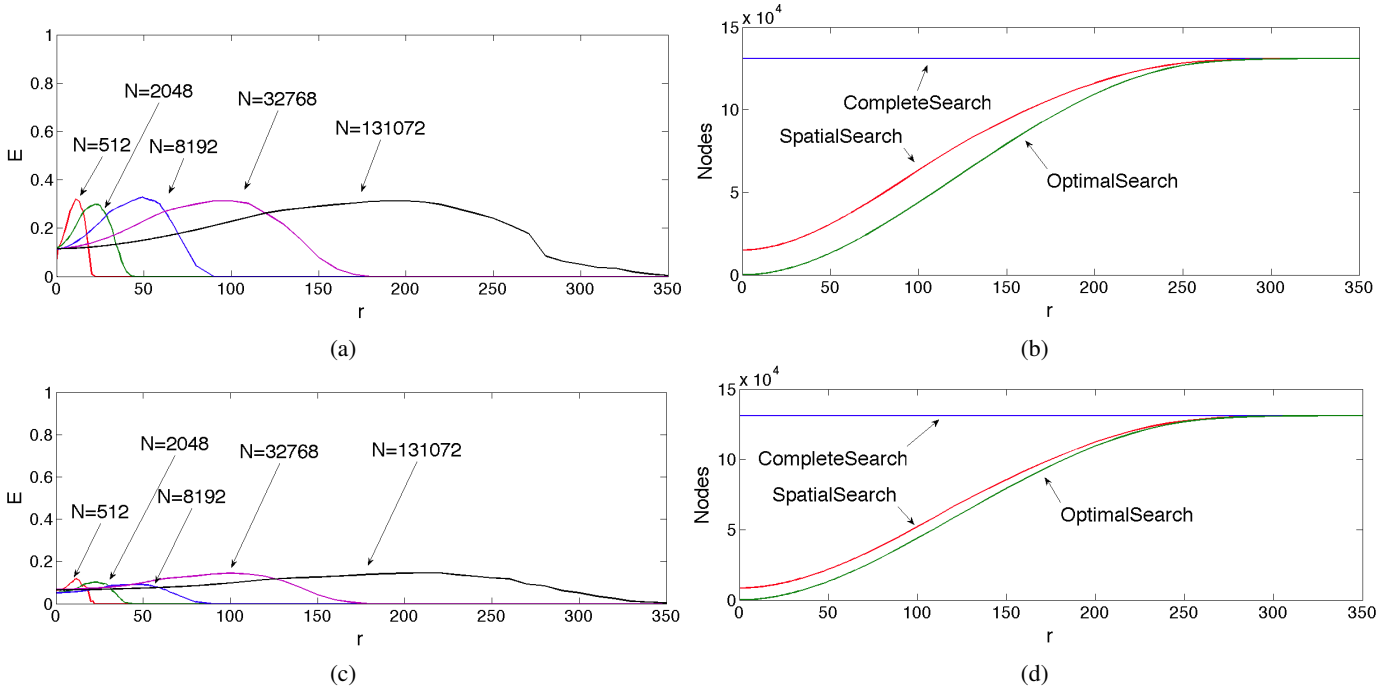


Fig. 4: The efficiency metric E of the SpatialSearch protocol for range(l,r) queries with location l and distance r. The top two plots show measurements on the *Barabási-Albert* topology, the bottom two on the *nearest-neighbor* topology. The plots on the left show efficiency vs distance, for different network sizes. $E = 0$ is the efficiency of OptimalSearch and $E = 1$ is that of CompleteSearch. Note that the vertical axis are of different scales. The plots on the right show number of search nodes vs distance, for networks of size 131,072.

is that of CompleteSearch. Figures 4(b) and 4(d) show curves of the number of search nodes vs distance r, for networks of size 131,072.

Each measurement point on a curve gives the average efficiency of 1000 queries, whereby each query is invoked with a start node and a location l, both of which are chosen uniformly at random on the network graph. The confidence intervals of the measurement points are too small to be shown in the figures.

A query with $r = 0$ corresponds to a search for objects with a given location l. Figure 4(a) shows that the efficiency of this query is about 0.12 for the Barabási-Albert topology for all tested network sizes. With increasing r, the efficiency increases to about 0.31 and then decreases to approach 0. For $r \geq \sqrt{2}s$ (s is the side of the square area, see above), the efficiency of the protocol becomes undefined and the performance of OptimalSearch, SpatialSearch, and CompleteSearch becomes the same. This can be seen in Figure 4(b), which shows the performance of SpatialSearch, measured in the number of locally searched databases, for a network of 131,072 nodes. The performance of the protocol on the nearest-neighbor topology, presented in Figures 4(c) and 4(d), is closer to OptimalSearch than on the Barabási-Albert topology. Otherwise, the protocol shows the same qualitative behavior as when it executes on the Barabási-Albert topology.

We performed similar evaluations for SpatialSearch with kNearestNeighbors query. For lack of space, the details are presented in [16].

B. Results

The described experiments allow us to draw the following conclusions. First, for range queries with small values of r (and kNearestNeighbors queries with small values of k), SpatialSearch is more efficient than CompleteSearch. In the case of large values of r (and large values of k), SpatialSearch performs close to CompleteSearch, which means that almost all network nodes need to be searched. Second, for the same value r, SpatialSearch performs better on a larger network than on a smaller network. (The same is true for the value k in kNearestNeighbors query.) Third, SpatialSearch performs significantly better on the *nearest-neighbor* topology than on the Barabási-Albert topology in terms of the number of nodes whose databases are searched. This is an artifact of the topology design, as explained above. For range queries, SpatialSearch performs close to OptimalSearch.

VII. Related Works

Euclidean spatial models have been used in different contexts, including geographic information systems, sensor networks, and network-aware overlays [18], [19], [20]. The first two types of systems use geographic coordinates. The last one uses so called network coordinates, which are virtual. Examples of network coordinate systems are Vivaldi [2] and Pharos [21]. In these systems, the Euclidean distance between

the locations of two nodes represents the end-to-end delay between them.

Spatial queries that are used with the above Euclidean spatial models include the *point* query, which returns objects at a given location, the *range* query, which returns all objects in a given spatial object, the *nearest-neighbor* query, which returns k objects nearest to a given location, and the *centroid* query, which returns the objects at the centroid of a given set of locations [22], [20].

A number of peer-to-peer (p2p) frameworks have been developed that maintain a distributed index for processing spatial queries. Some of these frameworks use Distributed Hash Tables (DHTs) for query routing [4], [5]. Scrap and PRoBe are two well-known examples of systems whose indexes are based on DHTs [4], [5]. They use the join and the leave operations of the DHT to construct and maintain the index in a decentralized way. In the case of Scrap, the multi-dimensional location space is mapped onto a single dimension in the form of a space-filling curve, i.e., the z-curve, which defines a single-dimensional identifier space that is managed by a DHT framework called skip graph [23]. The problem with this approach is that Scrap does not guarantee the preservation of locality in higher-dimensional location spaces. In our design, locality is always preserved.

Second, in the case of PRoBe, the multi-dimensional location space is directly mapped on to an identifier space of the same dimensionality, which is managed by the Content Addressable Network (CAN) [24]. While the PRoBe design preserves locality, query processing can still result in high overhead and high latency, for instance, when a peer invokes a point query for a location that is far away from its hyper-rectangle. In fact, due to the underlying CAN design, the latency for a point or a range query in PRoBE increases with $O(k\sqrt[k]{n})$ for a network of n nodes. In contrast, the latency of such a query in our design increases with $O(\log n)$ (given that the diameter of the network graph grows with $O(\log n)$), which is significantly smaller. This means that our design enables much faster query processing in large networks.

Apart from DHT-based indexes, other approaches have been developed to efficiently process spatial queries, first for centralised and later for distributed systems. The most widely used index structure is the R-tree [3]. During the last decade, designs have been devised to use R-trees in p2p frameworks that underlie large-scale spatial information systems [6]. A well-known example of a p2p framework that is based on R-trees is VBI-tree [6]. In this framework, the index is built starting from a given R-tree, which is then distributed to peer nodes, either by a human administrator or by using a centralized algorithm. Recently, a comprehensive p2p framework for processing spatial queries called MIDAS has been proposed [7]. MIDAS uses an index called k-d tree [25]. In the MIDAS design, the leaves of the k-d tree define the partitions of the location space, each of which is allocated to a peer. For query routing, each peer u has the knowledge of another peer v in each subtree for which there exists no larger subtree that contains v, but not u. The MIDAS design contains algorithms for point queries, range queries, and nearest-neighbor queries. Query latency increases with $O(\log n)$ for a network of n peers. Both VBI-tree and MIDAS is not suitable for our purpose, since the index structure in our case requires bottom-up construction.

VIII. Conclusion

We propose a peer-to-peer design for a spatial search system. In contrast to recent works, our design is bottom-up, which makes query routing network aware. The MBRIndex protocol creates and maintains a distributed index of object locations. It adapts to object and node churn. The index is build using the MBR concept, which allows to efficiently encode a large number of object locations. The SpatialSearch protocol prunes the search space and performs query routing. It supports a range of spatial queries, such as point, range, and k-nearest neighbor queries, which are implemented by realizing a set of abstract methods.

Our measure of protocol overhead is the number (or the fraction) of nodes whose local databases are searched during the execution of a query, and we compare the overhead of SpatialSearch with an (hypothetical) optimal protocol and a baseline protocol that searches all databases. Extensive simulations lead us to the following conclusions. First, SpatialSearch shows significantly lower (up to eighty percent) overhead than the baseline protocol, for point, range, and k-nearest neighbors queries. For the case of the point query, SpatialSearch performs close to the optimal protocol. Second, we investigated the scalability of the protocol for a system with up to 131,072 search nodes. We found that the efficiency of SpatialSearch, compared with the baseline protocol, improves with increasing network size, for all queries investigated. The larger the network size, the closer the performance becomes to that of the optimal protocol. Lastly, we found that the network topology has a major impact on the performance of the protocol. It is specifically efficient if neighboring nodes of the network topology have a small spatial distance.

With respect to execution time, SpatialSearch is asymptotically optimal by design. The simulations show no significant improvement of the response time for SpatialSearch compared with the baseline protocol, for all queries investigated. Further, point queries have shorter response times than any other types of queries.

Our plans include studying the impact of the churn rate on the protocol overhead and on the precision of query results for dynamic scenarios. We also plan to integrate spatial search into the framework of network search, which means extending the query language of network search, including a matching and ranking semantics for spatial queries, integrating the respective architectural designs, and producing an integrated prototype. Finally, we see interesting use cases when applying our approach to spatial search in the IoT technology domain.

References

[1] M. Uddin, R. Stadler, and A. Clemm, "Scalable matching and ranking for network search," in *9th International conference on network and service management. Zurich, Switzerland, 14-18 October 2013*.

[2] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," *SIGCOMM Comput. Commun. Rev.*, vol. 34, pp. 15–26, Aug. 2004.

[3] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, SIGMOD '84, (New York, NY, USA), pp. 47–57, ACM, 1984.

[4] P. Ganesan, B. Yang, and H. Garcia-Molina, "One torus to rule them all: Multi-dimensional queries in p2p systems," in *Proceedings of the 7th International Workshop on the Web and Databases: Colocated with ACM SIGMOD/PODS 2004*, WebDB '04, (New York, NY, USA), pp. 19–24, ACM, 2004.

[5] O. Sahin, S. Antony, D. Agrawal, and A. El Abbadi, "Probe: Multi-dimensional range queries in p2p networks," in *Web Information Systems Engineering (WISE 2005)*, vol. 3806 of *Lecture Notes in Computer Science*, pp. 332–346, Springer Berlin Heidelberg, 2005.

[6] H. Jagadish, B. C. Ooi, Q. H. Vu, R. Zhang, and A. Zhou, "Vbi-tree: A peer-to-peer framework for supporting multi-dimensional indexing schemes," in *Data Engineering, 2006. ICDE '06. Proceedings of the 22nd International Conference on*, pp. 34–34, April 2006.

[7] G. Tsatsanifos, D. Sacharidis, and T. K. Sellis, "Index-based query processing on distributed multidimensional data," *Geoinformatica*, vol. 17, pp. 489–519, July 2013.

[8] D. Papadias and Y. Theodoridis, "Spatial relations, minimum bounding rectangles, and spatial data structures," tech. rep., University of California, San Diego, 1994.

[9] Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis, *R-Trees: Theory and Applications (Advanced Information and Knowledge Processing)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[10] D. Peleg, *Distributed Computing: A Locality-sensitive Approach*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000.

[11] M. Dam and R. Stadler, "A generic protocol for network state aggregation," in *In RVK 05, Linkoping, Sweden, June 14-16, 2005*.

[12] S. Dolev, A. Israeli, and S. Moran, "Self-stabilization of dynamic systems assuming only read/write atomicity," *Distributed Computing*, vol. 7, no. 1, pp. 3–16, 1993.

[13] G. Tel, *Introduction to Distributed Algorithms*. New York, NY, USA: Cambridge University Press, 1994.

[14] R. Stadler, "Protocols for distributed management," Tech. Rep. 2012:028, KTH, Communication Networks, 2012. QC 20120604.

[15] A. Montresor and M. Jelasity, "PeerSim: A scalable P2P simulator," in *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, (Seattle, WA), pp. 99–100, Sept. 2009.

[16] M. Uddin, R. Stadler, and A. Clemm, "A peer-to-peer design for spatial search system," tech. rep., ACCESS Linnaeus Center, KTH Royal Institute of Technology, 2015.

[17] R. Albert and A. Barabási, "Statistical mechanics of complex networks," *Rev. Mod. Phys.*, vol. 74, pp. 47–97, Jan 2002.

[18] K.-t. Chang, *Introduction to Geographic Information System*, ch. Chapter 2: Coordinate Systems, pp. 18–40. McGraw-Hill Internation Edition, 2006.

[19] R. I. Da Silva, D. F. Macedo, and J. M. S. Nogueira, "Spatial query processing in wireless sensor networks - a survey," *Inf. Fusion*, vol. 15, pp. 32–43, January 2014.

[20] B. Donnet, B. Gueye, and M. A. Kaafar, "A survey on network coordinates systems, design, and security," *Communications Surveys Tutorials, IEEE*, vol. 12, pp. 488–503, Fourth 2010.

[21] Y. Chen, Y. Xiong, X. Shi, J. Zhu, B. Deng, and X. Li, "Pharos: Accurate and decentralised network coordinate system," *Communications, IET*, vol. 3, pp. 539–548, April 2009.

[22] B. Wong, A. Slivkins, and E. G. Sirer, "Meridian: A lightweight network location service without virtual coordinates," *SIGCOMM Comput. Commun. Rev.*, vol. 35, pp. 85–96, Aug. 2005.

[23] J. Aspnes and G. Shah, "Skip graphs," *ACM Transactions on Algorithms*, vol. 3, p. 37, November 2007.

[24] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," *SIGCOMM Comput. Commun. Rev.*, vol. 31, pp. 161–172, Aug. 2001.

[25] J. L. Bentley, "K-d trees for semidynamic point sets," in *Proceedings of the Sixth Annual Symposium on Computational Geometry*, SCG '90, (New York, NY, USA), pp. 187–197, ACM, 1990.