

BORA: Routing and Aggregation for Distributed Processing of Spatio-Temporal Range Queries

Goce Trajcevski* Hui Ding Peter Scheuermann†
 Department of EECS
 Northwestern University
 Evanston, IL, USA
 {goce,hdi117,peters}@eecs.northwestern.edu

Isabel F. Cruz‡
 Department of CS
 University of Illinois at Chicago
 Chicago, IL, USA
 ifc@cs.uic.edu

Abstract

This work tackles the problem of answer-aggregation for continuous spatio-temporal range queries in distributed settings. We assume a grid-like coverage of the spatial universe of discourse, in which each cell is governed by a Base Station (BS) that communicates with the mobile users in its zone, and is also equipped with a server that has Moving Objects Database (MOD) capabilities. The MOD server stores the data for the moving objects in a given cell, processes the continuous queries pertaining to that cell, and is connected to the MOD servers in the neighboring cells. We demonstrate that, when a range query that spans over more than one cell needs to have its answer computed for a user located in a particular cell, by intelligently combining the transmission and the aggregation of the partial results, substantial improvements can be achieved at the global level. Towards this end, we present the BORA (Bresenham-based Overlay for Routing and Aggregation) tree, which is used to combine the transmission and local data aggregation along the routes to the destination of the query's answer.

1. Introduction and Motivation

Typically, in cellular settings, a particular Base Station (BS) is in charge of contacting the mobile users that are in its area of coverage. However, the actual locations of the objects are managed by (possibly) a hierarchy of servers and the choice of the architecture impacts the trade-off between the cost of *lookup* of a given user vs. the cost of *updating* the location [20]. Majority of the works in Moving Objects

Database (MOD) research and practice have focused on the efficient storage and retrieval of spatio-temporal data and processing various (continuous) queries, however, a typical assumption in most of them is the existence of a centralized server [9, 16, 15, 24]. Some recent works have considered distributed settings, in which part of the responsibility for maintaining the answers of the spatio-temporal queries is delegated to the mobile users [6, 7]. Adding this extra level of query-awareness to the dead-reckoning update policy (c.f. [25]) yielded substantial savings in the communications. However, even in these settings, the MOD-like capabilities are assumed to be centralized.

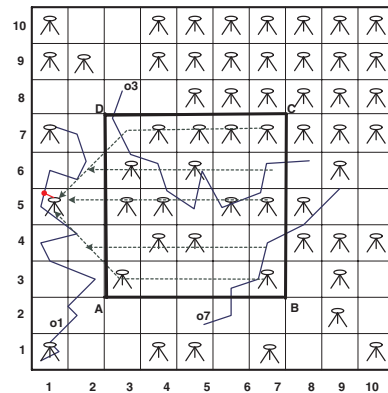


Figure 1. Problem Settings

In this work, we take a step towards a natural augmenting of a typical cellular architecture with a set of MOD servers. As a first approximation, we assume that each BS is coupled with a MOD, and that the servers are inter-connected in a mesh-like fashion. Each individual MOD is in charge of storage/retrieval of the mobile entities within the zone of coverage of the accompanying BS, and processes the spatio-temporal queries pertaining to its local region. An illustration of the settings that we assume is provided in Fig-

*Research supported by the Northrop Grumman Corp. grant PO 8200082518

†Research supported by the NSF – IIS-0325144/03

‡Research partly supported by the NSF – ITR: IIS-0326284 and IIS-0313553

ure 1, where the rectangle $ABCD$ indicates the region of interest R . Assume that at time $t = 10 : 00AM$ the following range query is posed:

Q1: Retrieve all the objects that will be inside R between 11:00AM and 11:20AM” (optionally, the user may request a specific time for the answer to be delivered, e.g., 10:40AM).

One observation is that the user would need to see the answer in some format like: $\{(o_{i1}, [t_{i1}^1, t_{i2}^1]), (o_{i2}, [t_{i2}^1, t_{i2}^2]), \dots, (o_{in}, [t_{in}^1, t_{in}^2])\}$, possibly even sorted by the objects ID (OID) attribute. The answers which are obtained by the individual MOD servers are relevant only for their cells and the query region may span over multiple cells. Moreover, the trajectories in the answer-set may also span over multiple cells throughout the time-interval of interest for the query, as illustrated by o_3 and o_7 in Figure 1. In such cases, if the final recipient is expected to do the aggregations of the multiple time-intervals per individual OID , it may experience too large workload, especially if the answer is to be sorted by the $OIDs$. In case the answer is to be delivered to an on-board device (e.g., a PDA) of a mobile unit, e.g. o_1 in Figure 1, the situation is even worse, both in terms of processing¹, as well as unnecessary exploitation of the wireless communication channel. Hence, it is desirable that, by the time the BS that controls the region in which the recipient of the query’s answer is located ($BS_{1,5}$ in Figure 1) starts its transmission, the corresponding server has already properly aggregated the answer-set, in the sense of merging the time-intervals of relevance for each individual OID and, if needed, sorting by the $OIDs$.

The main problem that we address in this paper is how to efficiently aggregate the final answer for a given spatio-temporal query, where the efficiency is measured in terms of the time that elapses from the moment that the individual servers begin to communicate with each other (after each of them has obtained its local answer), until the moment the particular server whose BS is in charge of transmitting the answer has all the partial answers properly aggregated. At the heart of our motivation is the following observation: Since a trajectory that is part of the answer may span over multiple cells, and the server of each cell needs to have its local answer transmitted along a path which involves neighboring cells, the overall time spent on aggregating the answer may be improved if some partial aggregation is performed along the way. Furthermore, if the final answer is also to be sorted by the $OIDs$, and assuming that each MOD server, besides the spatio-temporal indexing structures, e.g. [23], also has the local $OIDs$ sorted, then letting the destination server do the sorting of all the data from local partial results may be way too costly. Namely, this would amount to (an instance of) a k -way merge and, assuming that in each of the k sets we have m elements, the

complexity is $O(km \log m)$ if a heap/priority queue is used ($O(k^2m)$ ”brute-force”) [10]. If no sorting by the $OIDs$ is required, the problem of aggregating time intervals is essentially a problem of duplicates-elimination which, e.g. with a global hashing, can be done in linear time. As an illustration, consider:

Example 1. Assume a 10×10 grid (c.f. Figure 1) in which query region R is a 5×5 rectangle $ABCD$, whose lower-left coordinate is at the cell $C_{3,3}$ and the upper-right one is at the cell $C_{7,7}$ (inclusive). Further, assume that the destination of the answer is the server in the cell $C_{1,5}$.

A naive’ approach to aggregating the answer would be to fully exploit the parallel transmission of the data and aggregate the partial answers at the destination server $C_{1,5}$. Possible routes for this are illustrated with the dotted-arrowed lines in Figure 1 and observe that, besides the destination, there will be a contention at the nodes $C_{2,4}$ and $C_{2,6}$. Assume that the average number of trajectories (equivalently, $OIDs$) per cell is 5,000 throughout the duration of the query (20min.). Also, assume that, on the average, a trajectory of a moving objects spans over 4 cells. If the server at $C_{1,5}$ receives the partial answer-sets of each individual server involved in processing of the **Q1**, then it will end up having to execute an aggregation of the time-intervals among 125,000 $OIDs$, and $OIDs$ are sorted, then it will need to execute a 25-way merge of (sorted) sets of size 5,000 each. However, if $C_{2,4}$ and $C_{2,6}$ begin to aggregate the data that they have received, while waiting for $C_{2,5}$ to finish the transmission, each of them will execute the time-intervals aggregation for 50,000 $OIDs$ (or 10-way merge of size 5,000 if sorting is required). An important benefit is that, since the trajectories span over several cells, once $C_{2,4}$ and $C_{2,6}$ are done with the aggregation, the total data that each of them will transmit to $C_{1,5}$ will be much less than 50,000 ($o_i, [t_{i1}, t_{i2}]$) items (in our experimental setup, we obtained that the answer sizes in $C_{2,4}$ and $C_{2,6}$ was approximately 25,000). Hence, now the server at $C_{1,5}$ will have to aggregate a total of 75,000 elements (or a 7-way merge; 5 of size 5,000 and 2 of size 25,000). Using our experimental setup (c.f. Section 4) we obtained that for the settings above, the total time for completing the aggregation of all the individual cells’ partial answers at $C_{1,5}$ was 2.4 sec. (12.5 sec. with sorting), and 1.1 sec. if the partial results were pre-aggregated in $C_{2,4}$ and $C_{2,6}$ (4.6 sec. with sorting). Consequently, an aggregation speed-up of a factor ≥ 2 can be achieved with performing some partial aggregation ”on the fly”. However, there is another benefit – savings in the transmission due to the smaller sizes of the data after partial aggregation.

Hence, it is desirable to have a routing strategy that will: (1.) enable some partial aggregation; (2.) retain a certain amount of parallelism, for the purpose of avoiding too frequent aggregation of the data. Intuitively, such structure

¹As an example, the TX PDA operating under GarnetOS, has a 312MHz processor, 128MB of a total memory (<http://www.pcworld.com>)

should be a tree rooted at the (location of the) destination server which will preserve to some extent the "locality" of the geometric shape obtained from the query region and the collection of the shortest paths towards the destination. Along these lines, the main contribution of this work are as follows:

- We introduce the BORA (Bresenham-based Overlay for Routing and Aggregation) tree, a structure that provides a balance of parallelism and aggregation.
- We provide algorithms for constructing the BORA tree and aggregating the answer-set for a range query evaluated by a distributed set of servers.
- We provide experiments which demonstrate that significant improvements of the time-efficiency can be achieved in comparison with the naive approach.

The rest of this paper is structured as follows. In Section 2 we recollect the preliminary background. In Section 3 we present our main results, the BORA tree and the algorithms for routing and aggregation of the answer, and in Section 4 we present our experimental observations. Finally, Section 5 positions our work with respect to the related literature, concludes the paper and outlines directions for future work.

2. Preliminaries

In this work, we adopt a *trajectory* as model for the objects' motion, which can be defined as a sequence of points (polyline): $(x_1, y_1, t_1), (x_2, y_2, t_2), \dots, (x_n, y_n, t_n)$ where $t_1 < t_2 < \dots < t_n$. At each t_i , the object is assumed to be at the location (x_i, y_i) and in-between two consecutive points, at time $t_{in} \in (t_i, t_{i+1})$, the location of the object is obtained by a linear interpolation along the segment $(x_i, y_i)(x_{i+1}, y_{i+1})$. To construct its future motion plan, an object transmits its start-location, start-time and end-location (plus a set of to-be-visited points) to the server. The server has a *map* available, describing the graph of the road-segments, as well as the information about the velocity-distribution patterns on each segment in a given time-interval. Given the starting and end-location information, the server uses a dynamic variant of the Dijkstra's algorithm [4] in which the cost of the edges in a graph (road segments) depend on the time, in order to generate a travel-time optimal trajectory, which is subsequently transmitted back to the user [24, 3]. In our settings, each MOD manages only the portion of a given object's trajectory which is in its cell – the region of coverage of the *BS* that can communicate with the object during the time it is inside the cell. Hence, the servers will need to utilize a distributed version of the dynamic variant of the Dijkstra's algorithm [22]. Efficient algorithms for processing many of the popular spatio-temporal queries for trajectories are presented in [12], which can be executed locally by the servers in each cell. Note that the work in this paper is applicable to both

past and *future* trajectories. Before we proceed with our main results, we need to elaborate on two more issues:

Bresenham Algorithm: Bresenham's line algorithm is an algorithm that determines which points in a raster should be plotted in order to closely approximate a line segment between two given points. It is commonly used to draw lines on a computer screen, as it uses only integer addition, subtraction and bit shifting, all of which are very cheap operations in standard computer architectures. It is one of the earliest algorithms developed in the field of computer graphics [5]. The typical assumption is that the end-points of the line segments have integer coordinates and that the centers of the cells, i.e., the *pixels* are also in located in the integer-valued locations of a 2D mesh. The question is: given two pixel-points $A(x_A, y_A)$ and $B(x_B, y_B)$, end-points of a segment \overline{AB} , how do we determine which one of the neighboring pixels to A should be illuminated? The slope of the line defined by A and B is $m = (y_b - y_A)/(x_B - x_A)$, and assume, for the time being, that $0 \leq m \leq 1$ and that, as usual, the positive orientation along the X-axis is left-to-right. Now, the decision that has to be made is, when moving to the next column of pixels (i.e., increase the value of x by 1, from x_A to $x_A + 1$) which pixel in that column should be illuminated. Given the assumption about m , it will be either the point with the *same* value for y ($= y_A$), or the value which is 1-greater ($= y_A + 1$). The choice is done by simply rounding the value of m to the nearest integer, and adding that value to y_A . Clearly, for the subsequent point/pixel ($x = x_A + 2$), the value of $2m$ will have to be rounded to the nearest integer and added to the previous value along the y -axis.

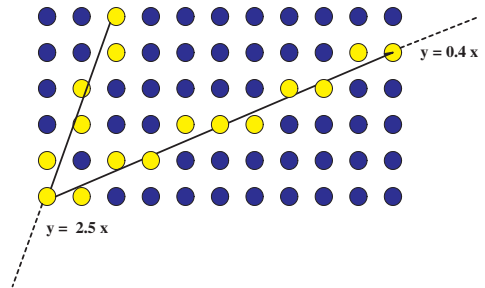


Figure 2. Bresenham's Algorithm

In case the slope is $1 < m < \infty$, then the procedure above has to be slightly modified, in the sense that one will certainly move progressively in the values along the y -axis, increasing the value by 1 in each subsequent column, however, the check will have to be performed whether the value along the x -axis should be increased by one or left unchanged, based on the rounding of $1/m$ to the nearest integer. Figure 2 illustrates the main aspects of the Bresenham's algorithm. Similar categorization applies when the slope of the line segment \overline{AB} is negative ($m < 0$).

Networking Aspects: As indicated in Section 1 (c.f. Figure 1), we assume a mesh-like connectivity among the servers in the cells, and the communication between two is connection-oriented, e.g., TCP-like [21]. However, for the actual throughput, besides the capacities of the communication link per se, in the actual TCP-like communication one needs to consider various factors, e.g., the round-trip time (RTT); the size of the receiver/sender window [21]; errors/packet-drops. As a particular example, the measurements reported in [14] indicate that, typically, 20% of the TCP flows have receiver windows limited to 8KB; 35% to 16KB and the rest 45% to 64KB. Throughout the rest of this paper, we will assume that the time-cost of the communication when transmitting n data packets between two hosts can be expressed as $b_1 + b_2 \cdot n$, where b_1 represents the round-trip time (ACK) for establishing the connection and b_2 represents the throughput, approximated by $receiving_window_size / round_trip_time$ [11]. Assuming that $b_1 = 320$ milliseconds and that the size of the window is $64KB$, we obtain that the throughput is $200KBps (= 1/b_2)$, and to observe the impact of these parameters, assume that each element of $(o_i, [t_1, t_2])$ takes 8B (24B total).

Example 2. In the context of Example 1, assume that, in case of a contention, $C_{2,5}$ gets the priority to transmit to $C_{1,5}$ (followed by $C_{2,4}$ and $C_{2,6}$). The total time-cost of the transmission for the partial answers to the query $Q1$ using the naive approach is $(9 \times 0.3) + (25 \times 5,000 \times 1/(200 \text{ KBps}) \times 24B) = 18.7 \text{ sec}$. Note that, past this point, the server at $C_{1,5}$ has to aggregate the entire collection of partial results, which (c.f. Example 1) yields 21.1 sec . (31.2 sec. with sorting). On the other hand, if the aggregation is used in $C_{2,4}$ and $C_{2,6}$ while waiting to connect to $C_{1,5}$, then the time-cost of communication is $((6 \times 0.3) + (2 \times 6 \times 5,000 \times 1/(200 \text{ KBps}) \times 24B)) + ((2 \times 0.3) + (2 \times 25,000 \times 1/(200 \text{ KBps}) \times 24B)) = 14.6 \text{ sec.}$, with smaller data sets to be aggregated both at $C_{2,4}$ and $C_{2,6}$ (in parallel), as well as at $C_{1,5}$ at the end. The overall time-cost to finalize the aggregation of the answer now becomes 15.7 sec . (19.2 sec. with sorting). Observe that, in case the network connection is faster (both smaller b_1 and larger b_2) the benefits of the partial aggregation will be more significant.

3 BORA-based Range Query Processing

Now we present the main result of our work – the algorithm for constructing the BORA tree and the algorithm for aggregation of partial answers which uses a given BORA tree as a routing structure.

For a given range query, we assume that the query-region of interest R , is an arbitrary polygon. The cells which intersect R are the ones which participate in the answer to the query. Let C_{x_d, y_d} denote the cell which is the destination (e.g., $C_{1,5}$ in Example 1) where the query's answer is

needed. Let $S_{i,j}$ denote the location of the MOD server of a cell $C_{i,j}$ and, in particular, S_{x_d, y_d} denote the location of the destination (assume they coincide with the cells' centroids). Also, assume that the pixels used in the Bresenham algorithm are located in the points with coordinates $S_{i,j}$, and the appropriate scaling is applied so that the distance between two consecutive points along each coordinate is 1. Let $C_R = \{C_{i_1, j_1}^R, C_{i_2, j_2}^R, \dots, C_{i_q, j_q}^R\}$ denote the set of cells which have a non-empty intersection with the query region R .

In order to find its relative position in the Bresenham-based tree for a given query region R , the server in each cell $C_{i,j}$ executes the following algorithm:

Algorithm 1:

1. If $C_{i,j} \in C_R$ OR ((the line through $S_{i,j}$ and S_{x_d, y_d} intersects a cell in C_R) AND (S_{x_d, y_d} is not between $S_{i,j}$ and the first intersection of that line with R))

- 1.1. Draw the line segment $\overline{S_{i,j} S_{x_d, y_d}}$
- 1.2. Among the neighbors $C_{(i+u), (j+v)}$ (where $u, v \in \{-1, 0, +1\}$), detect the pixel $S_{(i+u), (j+v)}$ that would be illuminated in the Bresenham algorithm for $\overline{S_{i,j} S_{x_d, y_d}}$
- 1.3. Select the server in the cell $C_{(i+u), (j+v)}$ to be the parent of the $C_{i,j}$ in the tree.

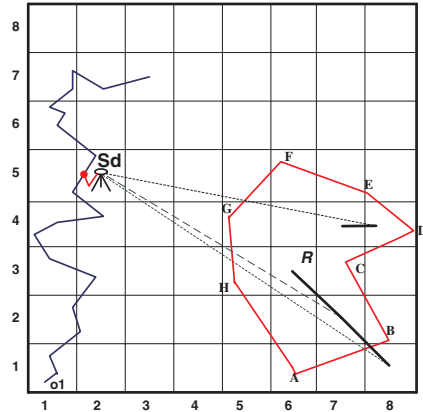


Figure 3. Constructing the BORA tree

Figure 3 illustrates the main aspects of Algorithm 1 for constructing the BORA tree, using a query region R represented by the polygon $ABCDEFGH$ and a destination point for the query's answer located at $C_{2,5}$. The dotted (and dashed) line segments indicate the inputs to the Bresenham's algorithm for the respective cells, and the thick lines indicate the corresponding branches of the BORA tree. Observe that the ancestry line that an individual cell would obtain using the native Bresenham algorithm need not coincide with the actual ancestry line of that cell in the BORA tree. The reason for this is that in line 1.3. of Algorithm 1, an individual "pixel" chooses only its parent. As an exam-

ple, observe the cell $C_{8,1}$ in Figure 3. Using the native Bresenham algorithm, the grand-parent node of $S_{8,1}$ towards $S_{2,5}$ would be $S_{6,2}$ of the cell $C_{6,2}$, as illustrated with the dotted line. However, in the BORA tree, its actual grand-parent is $S_{6,3}$ of the cell $C_{6,3}$ – the parent of $C_{7,2}$ (dashed line). Another interesting property² of the BORA tree is as follows. Let $V_R = \{v_1, v_2, \dots, v_k\}$ denote the vertices of the query region R . If S_{x_d, y_d} is a vertex on the convex hull of $V_R \cup \{S_{x_d, y_d}\}$, then the root has either three children, or five children (which only happens when the root is collinear with its two incident vertices on the convex hull). Otherwise, (S_{x_d, y_d} is in the interior of R or R is concave), the root may have up to eight children. However, an inner node still has up to three children.

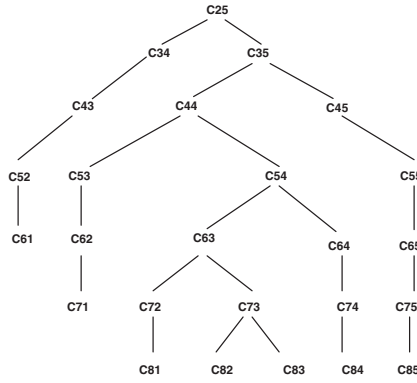


Figure 4. BORA Tree for the Region R

The BORA tree corresponding to the example range query depicted in Figure 3 is illustrated in Figure 4, where the nodes are labelled with the indices of the corresponding cells in the grid.

Now we proceed with describing the algorithm for routing and aggregating the partial results of a given range query, using the BORA tree. Let $LA(C_{i,j})$ denote local answer that the cell $C_{i,j} \in \mathcal{C}_R$ has calculated for a given range query. After it has been transmitted to its parent, call it $P(C_{i,j})$, for the first time, $LA(C_{i,j})$ is set to \emptyset . At any time, $C_{i,j}$ may have a local data-set relevant for the query, denote it $LD(C_{i,j})$ which is $LA(C_{i,j})$ aggregated with the data that it has received from its children – where $C_c(C_{i,j})$ denotes the c -th child of the $C_{i,j}$, in order from left-to-right in the BORA tree. We assume a `transmit-LD` function which formalizes the request from $C_{i,j}$ to transmit data to $P(C_{i,j})$ and, when granted (ACK), it sends all the data of $LD(C_{i,j})$ to $P(C_{i,j})$. Once a given node sends $LD(C_{i,j})$ to its parent $P(C_{i,j})$, the value of $LD(C_{i,j})$ is set to \emptyset , and no transmission can be requested whenever $LD(C_{i,j}) = \emptyset$. Now we have the following:

²Due to a lack of space, we do not make any formal "property/proof" statements.

Algorithm 2

1. *If $C_{i,j}$ has 0 children (leaf)*
 - 1.1. `transmit-LA`($C_{i,j}$) to $P(C_{i,j})$
2. *Else-If $C_{i,j}$ has one child*
 - 2.1. *If $C_1(C_{i,j})$ is ready to transmit*
 - 2.1.1. receive the data from $C_1(C_{i,j})$
 - 2.1.2. aggregate it with $LA(C_{i,j})$
 - 2.1.3. `transmit-LD`($C_{i,j}$) to $P(C_{i,j})$
3. *Else-If $C_{i,j}$ has 2 children*
 - 3.1. *If both children ready to transmit*
 - 3.1.1. receive data from $C_1(C_{i,j})$
 - 3.1.2. receive data from $C_2(C_{i,j})$
 - 3.1.3. aggregate the results with $LA(C_{i,j})$
 - 3.1.4. `transmit-LD`($C_{i,j}$) to $P(C_{i,j})$
 - 3.2. *Else-If one child, say $C_1(C_{i,j})$, is ready to transmit*
 - 3.2.1. execute "2.1" above with that child
4. *Else-If $C_{i,j}$ has three children*
 - 4.1. *If all three are ready to transmit*
 - 4.1.1. Let $C_2(C_{i,j})$ and $C_3(C_{i,j})$ aggregate their data at $LD(C_2(C_{i,j}))$
 - 4.1.2. Execute "3." above for $C_1(C_{i,j})$ and $C_2(C_{i,j})$
 - 4.2. *Else-If two children are ready to transmit*
 - 4.2.1. Execute "3." above for those two children
 - 4.3. *Else-If one child is ready to transmit*
 - 4.3.1. Execute "2.1" above with that child

Observe that each child $C_c(C_{i,j})$ can inform its parent when there is nothing else to be transmitted from it, with respect to the local results for the given query pertaining to its entire subtree. This information can recursively gathered in a bottom-up manner. Thus, Algorithm 2 is implemented to guide the transition of the states of the individual cells, until their role in the processing of a given query is completed.

A minor modification of the Algorithm 2 is needed for the special case of the root node. Namely, the root simply waits for all of its children to finalize their transmission, and then aggregates (merges) the collections of data that it receives. We reiterate that in each cell, the *OIDs* are sorted and that in the essence, the *aggregate* procedure actually merges its inputs so that it produces the output which is again sorted by the *OID* attribute and, in addition, when a particular *OID* is an element of more than one partial answer, it combines the time-intervals (union).

As an example, let us illustrate part of the execution of the Algorithm 2 for the BORA tree depicted in Figure 4 (in a top-down manner). Firstly, observe that $C_{2,5}$ will receive all the collections of partial answers along the path originating in $C_{6,1}$. Assume that $C_{4,4}$ is done aggregating $LA(C_{4,4})$ with the data along the path starting at $C_{7,1}$, before $C_{4,5}$ is ready for a transmission. Then, $C_{3,5}$ will aggregate the result received from $C_{4,4}$ with $LA(C_{3,5})$ and transmit it to the root. Subsequently, $C_{3,5}$ will forward the aggregated result that it receives from $C_{4,5}$ to the root. In a similar manner, the subtree rooted at $C_{5,4}$ will generate the data that will be forwarded to the root via $C_{4,4} \rightarrow C_{3,5}$. At the

end, the root $C_{2,5}$ will have to merge 8 different data sets (albeit, of a larger size) sorted by the $OIDs$ of its elements, instead of the original 24 data sets that would end up being transmitted by the naive approach.

4 Experimental Observations

To evaluate the benefits of the BORA tree based approach, we implemented a distributed query processing scheme in Java on a PC with Pentium IV 3.06GHz, 1G MB memory and Windows XP platform. For our experiments, we considered a geographic area of a size $40 \times 40 miles^2$ and we divided the area using a 20×20 grid structure (each BS is in charge of a cell region of $2 \times 2 miles^2$). The moving object data was generated using a modified version of the random way-point model [2]. We had a total number of 1,000,000 moving objects, and each one starts at a randomly selected position in the region of interest. In each subsequent time stamp, the object picks a random direction and moves at a speed randomly distributed between $12mph$ and $30mph$ until it stops. The average length of the trajectories is around 10 miles such that an average trajectory crosses 5 cells. Throughout the experiments, we used rectangles and octagons as query regions and we varied their size so that the covered area grows from 5% to 60% of the entire geographic zone of interests, in the increments of 5%, and for each value we averaged the results over 10 runs. The duration of each query was fixed at 30 min. We had three groups of measurements for each value of the area of the query's region:

- (1.) In the first group, the destination cell was at the same y -value (horizontally) as the centroid of the query's region.
- (2.) In the second group, the destination cell was in a diagonal direction from the centroid of the query region.
- (3.) In the third group of experiments, we actually generated a set of "stretched" rectangles and octagons, where the horizontal dimension was much smaller than the vertical one (while retaining the value of the area of the query's region). In each group, we measured the processing times for: (1.) Naive approach – the local data from the individual cells is transmitted "as is" to the destination, using as much parallelism as possible (contention was resolved based on "row-major" order). The destination cell then aggregates the entire collection; (2.) Our approach based on the BORA tree and using Algorithm 2.

We assumed the communication is carried out in a per-hop manner, where each server is capable of concurrently transmitting and receiving data, where the RTT takes 0.3 seconds and the single-link transmission capacity is 200KBps. For each individual cell that participated in the answer, we pre-computed its contribution. For the aggregation of the partial answers when $OIDs$ needed to be sorted, we implemented the variant of the k-way merge algorithm

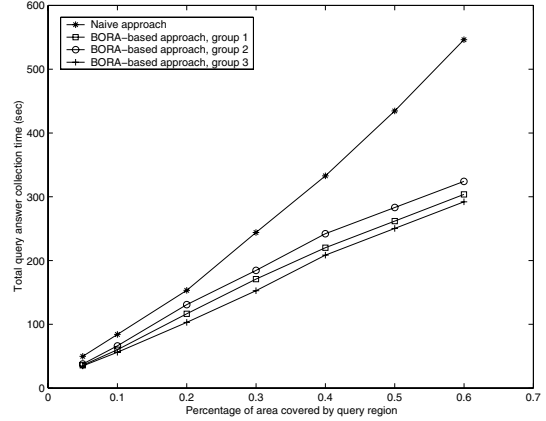


Figure 5. Naive vs. BORA (sorted $OIDs$)

which also properly aggregates the time-intervals for each moving object and measured the overall time.

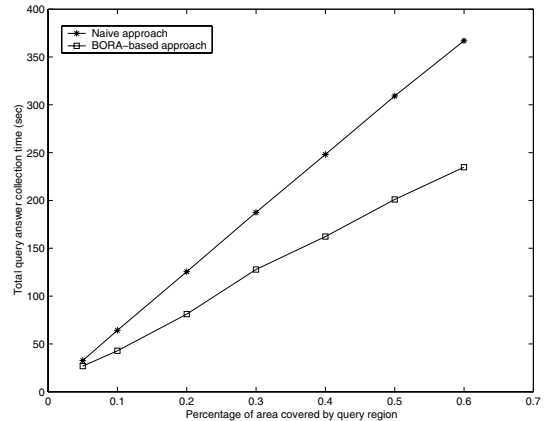


Figure 6. Naive vs. BORA (no sorting)

The total processing times for the different cases are illustrated in Figure 5 and they show that the BORA-based approach consistently outperforms the naive approach. The main savings in the processing time comes from two sources: (1) the distributed merging of the partial results; (2) the reduced size of query answers subsequently transmitted. The result of group 3 is slightly better than the other two groups since the BORA tree produced by the stretched query region has many internal nodes with 3 children, thus achieving better in-route aggregation.

We repeated the experimental setup and observed the performance of each approach for the case when the $OIDs$ were not expected to be sorted. The improvement trends are similar with the ones observed when a sorted aggregation was used, however, the respective values are smaller. This is due to the fact that when a simple partial aggregation is done, we still save on having smaller sizes for the

subsequent-hop transmissions, but the "in-site" processing costs of aggregating time-intervals only are not as high as when sorting by the *OIDs* was required. The results are depicted in Figure 6 and, as can be seen, we did not observe as significant discrepancies among the three groups for the BORA-based approach.

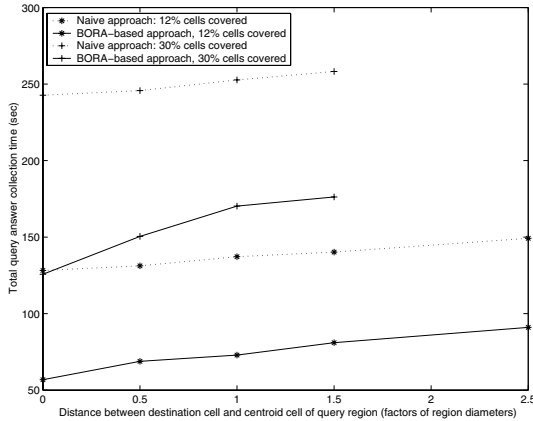


Figure 7. Destination Distance Impact

We also evaluated the impact of the distance between the destination cell and the (centroid of the) query region. For a given region, we increased the distance between the destination cell and the centroid of the query region, from 0 (the destination cell is at the centroid of the query region), to 0.5 times of the region diameter (the destination cell is close to the boundary of the query region), and up to 2.5 times of the region's diameter, away from the centroid. We averaged the results over 10 runs, using regions of sizes 12% and 30% of the total geographic area of interest. The experimental results comparing the naive approach and the BORA-based approach are shown in Figure 7. When the destination cell is at the centroid of the query region, the BORA-based approach yields a greater speed-up because there are a lot more internal nodes with 3 children, exploiting larger reductions of the data size during partial aggregations in parallel.

5 Related Work and Concluding Remarks

Much research has been done in the already established fields of parallel processing/algorithms [8] and distributed databases [18], compared to which the MOD research [9] is relatively young. In this work, we addressed some particular aspects of managing spatio-temporal range queries in distributed settings, thus bringing the peculiarities of MOD issues closer to these traditional fields, as well as the field of networking [21]. Specifically, we balanced the merging and transmission of the data, capitalizing on the

property that in the cell-based environment, a given trajectory may spread over more than one cell.

Recent works have addressed various aspects of the efficient processing of MOD queries for different motion models; - a sequence of $(location, time)$ updates [16, 26]; - a sequence of $(location, time, velocity)$ updates [25]. However, for the most part, the works assumed a centralized environment with respect to the MOD server. A work that is similar in spirit to ours is the MobiEyes project [6, 7]. The authors consider the problem of load balancing between a server and a set of mobile clients, along with the minimization of the communication overhead. The model of motion is the one of $(location, time, velocity)$ updates, and the standard dead-reckoning policy [25] is extended with techniques that maintain the correctness of the continuous queries. Our work is both orthogonal and complementary to [6, 7]. Firstly, we consider the motion model of a trajectory for the moving objects. Secondly, our problem considered a distributed set of servers and addressed the problem of efficient aggregation of the partial/local answers for the case of range queries.

Routing trees with the aggregation of the (partial) answers have been extensively used in sensor networks settings [1, 27]. Unlike ours, in these settings, the main purpose of the aggregation is to reduce the energy expenditure in the participating nodes, in order to prolong the networks' lifetime, while ensuring some quality of data guarantees. An approach which deals with the issue of the parallelism and concurrent transmission in sensor networks settings is presented in [28] and the work tackles the problem of minimizing the packets collision/drop for the purpose of energy-savings. We are also concerned with the issue of efficient balancing of transmission and partial results' aggregation, however, in completely different settings.

There is a plethora of works from the networking community that is related to ours, and can be further used to extend our results. We relied on [11, 14, 19], however, the ECO project [13] addressed the problem of analyzing a given network and establishing communication patterns for the purpose of developing efficient data-parallel applications. Although in this work we used simple model for the network infrastructure, the similarity with [13] is due to the fact that we also exploited the geometric properties of the queries, destinations, and the spatio-temporal data, all in order to provide efficient parallel aggregation of a distributed set of partial answers.

In conclusion, this paper presented an approach for efficient aggregation of the individual (local) answers for spatio-temporal range queries in distributed settings. Since the trajectories of the moving objects can span over more than one zone of coverage in a cellular environment, we observed that partial aggregations enable a reduction of the size of the data, thereby relieving the destination server

from merging too many data-sets. Based on the ideas of Bresenham's algorithm from computer graphics [5], we introduced the BORA tree as a routing structure, and we presented algorithmic solution for efficient balancing of the parallel transmissions and partial aggregations.

We believe that our work can serve as a starting point for several interesting research directions. As a first extension, we are planning to incorporate the other categories of spatio-temporal queries (e.g., k-NN, join) into our framework, and consider the combinatorial optimization problems of efficient routing and aggregation when multiple queries and various cell-loads are present. We are also planning to extend the work so that it can exhibit a reactive behavior, in the sense that it can efficiently incorporate the changes of the queries' answers due to traffic abnormalities which affect the trajectories that were used for generating those answers. It would be very interesting to incorporate the other motion models in the picture and extend the architecture in a manner that the set of distributed MOD servers is not in a 1-to-1 relationship with the set of the *BSs*. An ongoing project is to evaluate the Bora-based approach under a larger variety of networking parameters (c.f. [11, 14]). A particular long-term challenge is how to incorporate the dynamic geometric properties of the spatio-temporal data and the recent works on geometric models of the Internet for performance-aware protocols [17], for the purpose of efficient management of continuous spatio-temporal queries in distributed environments.

Acknowledgment: We thank Prof. Aleksandar Kuzmanovic for his constructive comments.

References

- [1] A. Boulis, S. Ganeriwal, and M. Srivastava. Aggregation in sensor networks: Energy-accuracy trade offs. In *SNPA*, 2003.
- [2] J. Broch, D. Maltz, D. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Mobile Computing and Networking*, 1998.
- [3] H. Ding, G. Trajcevski, and P. Scheuermann. Omcat: Optimal maintenance of continuous queries answers for trajectories. In *ACM SIGMOD*, 2006. (demonstration paper).
- [4] S. E. Dreyfus. An appraisal of some shortest – path algorithms. *Operations Research*, 17(3), 1969.
- [5] J. Foley, A. van Dam, S.K. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice in C*. Addison-Wesley, 1995.
- [6] B. Gedik and L. Liu. Mobieyes: Distributed processing of continuous queries on moving objects in a mobile system. In *International Conference on Extending the Database Technology (EDBT)*, 2004.
- [7] B. Gedik and L. Liu. Mobieyes: A distributed location monitoring service using moving location queries. *IEEE Transactions on Mobile Computing*, 5(10), 2006.
- [8] A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to Parallel Computing*. Addison-Wesley, 2003.
- [9] R. Güting and M. Schneider. *Moving Objects Databases*. Morgan Kaufmann, 2005.
- [10] D. Knuth. *The Art of Computer Programming, Vol.3: Searching and Sorting*. Addison-Wesley, 1998.
- [11] A. Kuzmanovic and E. Knightly. A performance vs. trust in the design of end-point congestion control protocols. In *ICNP*, 2004.
- [12] J. Lema, L. Forlizzi, R. Güting, E. Nardeli, and M. Schneider. Algorithms for moving objects databases. *Computing Journal*, 46(6), 2003.
- [13] B. Lowekamp and A. Beguelin. Eco: Efficient collective operations for communication in heterogeneous networks. In *IPPS*, 1996.
- [14] A. Medina, M. Allman, and S. Floyd. Measuring the evolution of transport protocol on the internet. *Computer Communications Review*, 35(2), 2005.
- [15] M. Mokbel, X. Xing, M. Hammad, and W. Aref. Continuous query processing of spatio-temporal data streams in place. *The GeoInformatica Journal*, 9(4), 2005.
- [16] M. Mokbel, X. Xiong, and W. Aref. Sina: Scalable incremental processing of continuous queries in spatio-temporal databases. In *ACM SIGMOD International Conference on Management of Data*, 2004.
- [17] T. Ng and H. Zhang. A network positioning system for the internet. In *USENIX*, 2004.
- [18] M. Ozsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 1999.
- [19] J. Padhye, V. Firooz, D. Towsley, and J. Kurose. Modeling tcp throughput: A simple model and its empirical evaluation. In *ACM SIGCOMM*, 1998.
- [20] E. Pitoura and G. Samaras. Locating objects in mobile computing. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 13(4), 2001.
- [21] K. Ross and J. Kurose. *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison-Wesley, 2004.
- [22] M. Sato and M. Nakamura. An efficient route generation algorithm for distributed multi-layered network. In *ISCC*, 1998.
- [23] Y. Tao, D. Papadias, and J. Sun. The tpr*-tree: An optimized spatio-temporal access method for predictive queries. In *VLDB*, 2003.
- [24] G. Trajcevski, O. Wolfson, K. Hinrichs, and S. Chamberlain. Managing uncertainty in moving objects databases. *ACM Transactions on Database Systems*, 29(3), 2004.
- [25] O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, 7, 1999.
- [26] X. Xing, M. Mokbel, W. Aref, S. Hambrusch, and S. Prabhakar. Scalable spatio-temporal continuous query processing for location-aware services. In *International Conference on Scientific and Statistical Database Management (SSDBM)*, 2004.
- [27] M. Youssef, M. Younis, and K. Arisha. A constrained shortest-path energy-aware routing algorithm for wireless sensor networks. In *IEEE-WCNC*, 2002.
- [28] V. Zadorozhny and P. Chrysanthis. Network-aware wireless sensor data management. In *MDM*, 2006.