CAWA: Continuous Approximate Where-About Queries

Alexander Aved, Kien A. Hua, and Antoniya Petkova

School of EECS
4000 Central Florida Blvd
Orlando, Florida, USA
{aaved, kienhua, apetkova}@eecs.ucf.edu

Abstract. With the current proliferation of wireless networks and mobile device technologies, the management of moving object databases to facilitate queries over these domains has been extensively studied. While current systems are concerned with tracing the precise location, or paths, of these objects, in many cases it is sufficient to know that the approximate location or path of a moving object is, with certainty, within known bounds. Thus, we present our work on Continuous Approximate Where-About (CAWA) queries, in which mobile sensors sense the presence of an object (not its precise location) and query servers deduce the object's where-about area from mobile sensor presence updates.

Keywords: Moving object database, continuous query, presence, distributed computing, where-about.

1 Introduction

Given the advances in scaling down the power consumption requirements for and miniaturization of processing devices, and with concurrent gains in processing power, we are becoming a society with massive numbers of mobile devices. Introduce communication mediums, such as the various forms of wireless networks, e.g., Mobile Ad Hoc Networks (MANETS) [1] and MESH networks [2], and it then becomes possible to communicate with and track these mobile devices. Inexpensive GPS receivers facilitate the devices' location awareness in terms of time and location. This mobile, location aware infrastructure provides many exciting applications: military planners are interested in knowing the location of a supply convoy in relation to nearby army tanks and known enemies. Parents waiting for a child to come home would be interested in knowing if she's still in the grocery store, and a courier service might be interested in rerouting vehicles located on a busy highway in which an accident has just been reported. In many instances, the precise location of a Moving Object is extremely relevant: two meters with respect to a car could be the difference between driving on a road and driving into a river beside the road. However, in many applications, an exact location is not required. It is sufficient to know that the location of the queried object is certain to be within a threshold, or more precisely, to know

with complete certainty that the exact location of the object is within a known region. Thus, it could be sufficient to know that the convoy is currently contained within a friendly area, that the child is somewhere within the grocery store, or that an RFID-tagged cargo container was detected on a truck when it passed an inspection station [10, 11, 12, 13, 14].

Shipping companies have thousands of delivery vehicles and move millions of packages through their delivery networks. Since it is not feasible to communicate with every object to resolve the location of a single one, we utilize moving object databases to coordinate the querying and location awareness efforts. Generally, queries can be point queries (what is the precise location of cellular telephone *X*), range queries (which trucks are currently within a certain range of a store), or predication queries (at what intersection will Taxi *Z* arrive in 15 minutes). Moving object locations can be known to the moving object database through location updates from the object itself, calculated trajectories as a function of time, and other probabilistic methods [15]. However, with current technology, knowing the precise location of a moving object is an extremely difficult problem, because once a location update is broadcast, the object may have already moved or there may be a latency introduced in the propagation of the location update on the physical transmission medium, etc. If there are millions of tracked objects, the moving object database server(s) must cope with millions of periodic location updates, and must cope with precise location uncertainty and either pass it along in the query result or utilize some probabilistic solution. In this paper we examine a new paradigm for moving object location tracking, which differs from the previously described scenario in two primary ways. First, we point out that in many applications, it is sufficient to know the approximate location of the moving objects within a bound. Second, by utilizing a network of sensors that detect the presence of moving objects (as opposed to receiving location updates from each object), we construct an environment that scales with the number of queries and is robust to the density of the tracked moving objects. Thus, we present our work on *Continuous Approximate Where-About* (CAWA) queries, which are based on mobile sensors that can detect the **presence** (but not the exact locations) of target moving objects as detected by mobile sensors. We present a system design that is both cost effective and scalable. Current design approaches are expensive because the system cost is proportional to the number of moving objects, as each moving object must include GPS and communication hardware to locate and report its locations. Our goal is to design a system whose cost is not dependent on the number of mobile objects, and to therefore substantially reduce the cost of implementation. To have a system that is both simple and scalable, we avoid using spatial indexing techniques, which have a logarithmic computation complexity with respect to the number of moving objects, and instead utilize a design based upon sequential data scans.

The remainder of this paper is organized as follows. We discuss related work in Section 2. The proposed query processing technique for CAWA queries is introduced in Section 3, with the distributed design presented in Section 4. In Section 5, we describe the simulation setting and examine the simulation results. Finally, we offer our conclusions in Section 6.

2 Related Work

The construction of databases for tracking moving objects is a widely researched topic. [16, 18, 19, 20, 21, 22] Due to the continuous movement intrinsic to the problem, the exact locations of the objects can only be known when they are sampled, and by the time that update becomes known to the server the moving objects have likely moved [3]. There are a great diversity of approaches to cope with this problem, for example, by restricting the type of queries that can be resolved by (1) utilizing logic to approximate and cope with error [4], or (2) restricting the movement of the moving objects themselves (e.g., some approaches limit the moving objects to a fixed topology, such as roads, to make the objects' movement easier to predict [5] with lower error or to reduce location updates).

Various architectures for moving object databases have been proposed over the years (e.g., [6]). Even with various indexing schemes (e.g., [7, 8, 9]), most designs are centralized in nature, and do not scale well as the number of moving objects increase. The way data is stored on the server, and the types of queries they can process effects their scalability.

2.1 Spatio-Temporal Queries

We identify two primary types of spatio-temporal queries: point queries (also called coordinate-based queries) and trajectory-based queries [16, 17]. Point queries return all relevant moving objects that satisfy a spatial relationship, e.g., the current location of all city busses. Trajectory queries refer to the trajectory of a single (or multiple) moving object for a time interval specified in the query, and provide results such as the area in which the moving object has traversed, how far it has traveled (a segment length), or just the set of moving objects that satisfy the query (e.g., which trucks traveled on Highway 99 in the past 2 hours).

2.2 Probabilistic Approaches

Due to the continuously moving and possible random nature of moving objects, it is not possible to precisely and continuously monitor and record their location due to the discreteness of measurement instruments. Thus, the query result returned by a moving object database may not be completely consistent with the real-world situation [23, 24]. The goal of probabilistic approaches is to extend the query language with semantics for dealing with this uncertainty. For example, "Which whales have been in Region 1 with probability 90%." [29]

There are two primary approaches for dealing with the representations of movement information: uncertainty about future behavior [24, 25, 26] and uncertainty about previous behavior [23, 27]. Approaches in the first category make use of speed and trajectory information to model the movement of the tracked moving objects in the database, in order to reduce the number of location updates. Examples abound, such as virtual digital battlefields, fleet management, etc. The second approach concerns

historical movements, compensating for uncertainty in the precise location of the moving object at a particular time. This has applications in data mining, pattern classification and recognition, and so forth.

3 CAWA Query Processing

A CAWA query is a continuous query that tracks the locations of a moving object to determine its whereabouts in the past *t* time units, where *t* is referred to as the *tracking window*. In this paper, the whereabouts of the object are represented by a *minimum bounding rectangle* that encompasses the locations of the query object in the past *t* time units. At any moment in time, this minimum bounding rectangle is the result of the query; and this result continues to change in size and shape over the course of the continuous query. In this section, we discuss a technique for processing CAWA queries in a single-server environment. We will present a distributed design in Section 4.

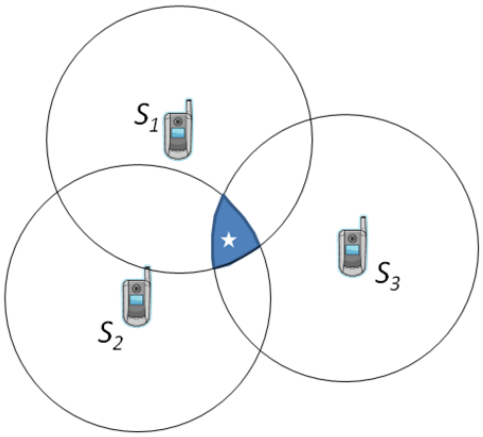

Fig. 1. A moving object (denoted as a star) is detected by three nearby mobile sensors. The solid area can be deducted as the where-about of this moving object at this moment.

There are two kinds of moving objects in the proposed environment: regular moving objects and mobile sensors. We assume that each mobile sensor is equipped with a GPS (*global positioning system*), and always knows its location. A mobile sensor continuously senses the presence of moving objects currently within its sensing range, and reports their identifiers (ID's) to the Query Server, along with the location of the sensor. As an example, a phone company can recruit cellular phones with integrated RFID (*Radio Frequency Identification*) readers to detect objects tagged with RFIDs, within the sensing range of the reader. In our environment, a mobile sensor can detect the identifiers (*ID's*) of the moving objects within its sensing range. However, the sensor does not know the exact locations of these moving objects.

Nevertheless, the query server can fuse information from multiple sensors to deduct the where-about of a moving object. As illustrated in Figure 1, the sensing range of each of the three mobile sensors S_1, S_2, and S_3, is a circular area; and the moving object is shown as a star at the center of the figure. If S_1 is the only sensor that detects this moving object, the best we can say is that the object is somewhere within the circular sensing range of S_1. However, since all three sensors S_1, S_2, and S_3, detect this moving object, the query server can safely deduce the where about of the moving object to the overlap area of the three sensing ranges (i.e., the shaded area in Figure 1). Unlike existing techniques that can process queries based on the locations of moving objects, we must rely on the location deduction mechanism in the CAWA environment. We note that triangulation techniques that can more "precisely" locate the location of a moving object exist (e.g., [31]). However, these schemes are computationally intensive (e.g., solving a nonlinear system of equations for each moving object) and seem better suited for robotics applications which typically deal with only a handful of moving objects.

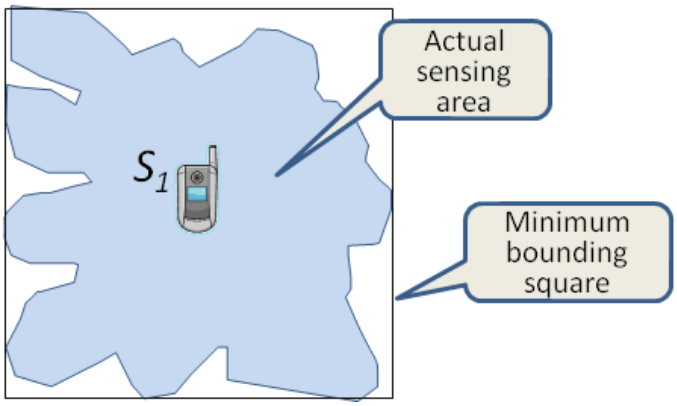

Fig. 2. Approximating the sensing area with a minimum bounding square.

One major advantage of the CAWA approach is the significant reduction in communication cost since only the mobile sensors (not every moving object) need to communicate with the query server. We will observe in the simulation results, presented in Section 5.2, that the number of mobile sensors required to achieve good performance is proportional to the area of the terrain, not the number of moving objects. This implies that the CAWA approach can be very economical to support a very large user community (i.e., a very large number of moving objects). In order to keep the computation cost low, we approximate the sensing range of each mobile sensor as the *minimum-bounding square* (MBS) of its sensing area. We note that the actual sensing area might have an irregular shape depending on the broadcast pattern of the antennas (see Figure 2). Using an MBS as the approximation is also a natural fit for computing the result of a CAWA query, which has a rectangular shape. Using MBS's as the sensing ranges, the where-about of a moving object can be approximated as a rectangular area as illustrated in Figure 3.

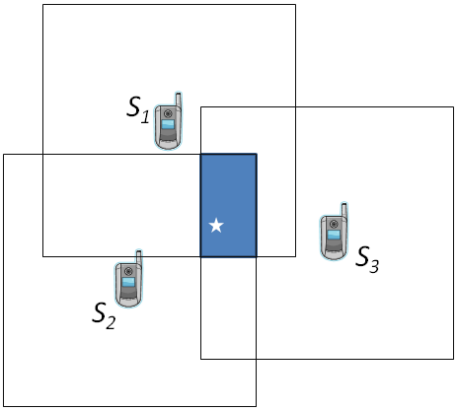

Fig. 3. Sensing range is approximated as a square area. A moving object (denoted as a star) is detected by three mobile sensors. The highlighted overlap area can be deduced as the where-about of the moving object at this moment.

The user poses a CAWA query Q to the query server by specifying the ID of the target moving object that is to be tracked. We refer to this target object as the query object. At any moment in time, the query object is detected by zero or multiple mobile sensors. If no mobile sensor detects the query object, its where-about for this moment in time is the entire terrain. Typically, the query object is detected by multiple mobile sensors; and the where about of the query object can be narrowed down to the spatial intersection of the areas of coverage of these mobile sensors. Let R(T) denotes this overlap areas at time T. Over the course of this continuous query, this overlap area changes in shape and size as the query object moves and it is detected by different mobile sensors at different times. The result of the CAWA query Q at time T can be computed as follows:

$$Q(T) = \bigcup_{t=T-W}^{T} R(t) \quad \textbf{(1)}$$

where the notation $\cup$ denotes a spatial union of the overlap areas R(t), $T\text{-}W \leq t \leq T$. W is the tracking window. That is, we are interested in the whereabouts of the query object in the last W time units.

To support CAWA queries, the query server maintains the following information:

- **LocTable**: This table maintains the current locations of the mobile sensors. That is, for each *sensorID*, we record its location.
- **QueryTable**: This table maintains the list of query objects (i.e., *objID*'s).
- **DetectedObjTable**: This table maintains the query objects detected by each mobile sensor. Each sensor also reports non-query objects. However, only query objects are recorded in this table.

- **ResultTable**: This table maintains the query results for each of the last T_{max} time units (i.e., the $R(t)$ discussed previously), where T_{max} is the maximum tracking window allowed for any query.

For each iteration, the query server performs the following procedure:

1. Receive data from mobile sensors and update the LocTable.
2. Also use the sensor information and information in the QueryTable to update the DetectedObjTable.
3. Update the ResultTable using information in the DetectedObjTable
4. Compute the result for each query by applying Equation (1) to the entries for each query recorded in the ResultTable.
5. Report the current results to the users.

The above procedure is repeated as long as there is at least one active query in the system. We note that some degree of uncertainty in the locations of the mobile sensors exists due to the iterative nature of the location update process. The validity of the reported locations decreases until the next round of location updates. This issue has been resolved using location estimation techniques (e.g., [28]). Another solution to reduce this effect is to use distributed servers. We present one such solution in the next section.

We discuss the details of the above procedure as follows:

- Step 1: This step can be done on the fly as location information arrives from the mobile sensors.
- Step 2: Since the QueryTable is typically small and fits in the memory, this step can also be done efficiently in parallel with Step 1.
- Step 3: We can scan the table DetectedObjTable, and group the entries according to query objects as follows:

```
SELECT   objID, sensorID
FROM     DetectedObjTable
GROUP BY objID
```

  We can then scan the result of the above group-by query, and for each query object (i.e., objID) compute the spatial intersection of the sensors involved. To speed up this process, a hash index on the sensorID attribute of the LocTable can be used.
- Step 4: If we sort the ResultTable according to the objID attribute (i.e., query objects), this step can be done in a single scan over this table.

Thus, the query processing technique is quite efficient, such that it involves mostly sequential scans of the data files. Only LocTable needs to be accessed in a random manner in Step 3. However, a hash index can be used to reduce the disk access cost.

4 Distributed Design

In the distributed scenario, the terrain is partitioned into multiple sub-domains, each of which has assigned a query server. Each query server is responsible for moving objects detected within its sub-domain. Each query server is also responsible for the processing of the where-about queries corresponding to the query objects currently within its sub-domain (see Figure 4).

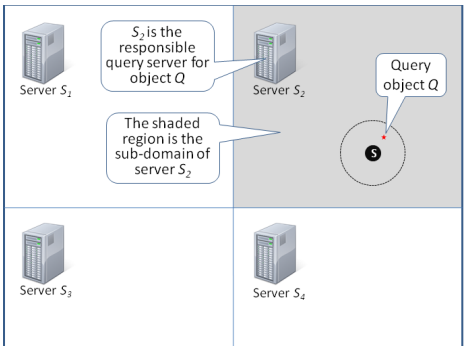

Fig. 4. This figure shows the terrain segmented into four sub-domains. Each sub-domain is assigned a server. Server S_2 is responsible for Query object Q.

Queries may be poised to any query server on the terrain. Upon receiving a new query from a user, this initial query server examines its local LocTable to determine if the query object is currently within its sub-domain. If this is the case, this query server declares itself the assigned query server for this query; otherwise, this initial query server broadcasts the query to all of the other query servers in the terrain to locate the appropriate query server for this new query. At this time, each of these query servers checks its own local LocTable to verify if it has the query object in its sub-domain. The query server that has the query object in its sub-domain can then take ownership of this particular new query.

The query server, responsible for a given query, processes this query using the technique described in Section 3. As part of this process, the query server tracks the current approximate location of the query object (i.e., its where-about rectangular area). When this approximate location is within a threshold of a neighboring sub-domain, the responsible query server forwards a copy of the query to the query server, say *S*, of the appropriate neighboring sub-domain. Upon receiving information on this query, the server *S* activates a countdown counter and anticipates the crossover of this query object before the counter expires. If the query object does cross over and be detected before the timeout, termed *elevation time out* (ETO), *S* is elevated to be the new responsible query server for this particular query; otherwise, *S* can discard the expired information on this query when the counter decrements it value to zero.

To support CAWA queries in a distributed environment, the following information is maintained on the query server:

- A queue of CAWA queries the server is actively processing. Each CAWA query consists of (1) the ID of the target object to be tracked (objID), (2) the ID of the user who submitted the query (who can access the query's results) and (3) an *active time span* (ATS) for which the query active (this time span is specified by the user when submitting the query and can be null, in which case the query is valid until explicitly terminated), (4) a unique ID for the query, and (5) the ETO (described previously in this section).

- A queue of the mobile sensors on the terrain, to permit the query server to track the sensors' states: (1) if the mobile sensor can be communicated with (is on-line and the query server has received a message from the mobile sensor within a time threshold) and (2) if it is currently tracking any moving objects that are the target of any queries.

5 Performance Study

To assess the performance of CAWA we compare the approximate where-about area for a moving object, the query result returned by the query server, with the ground truth, in terms of precision and recall. The precision metric provides the ratio of the area of the queried mobile node's exact movement (as contained in a minimum bounding rectangle) in terms of the query result (the approximate where about area) returned from the query server, for a window of previous positions. The recall metric provides indication of the area of the true positive area in terms of the false negative and true positive areas (Figure 5, Figure 6, and Table 1).

Table 1. Measurement metrics and corresponding descriptions

Metric	Description
TP, true positive	The overlapping area between the approximate (where-about) area and the exact area (ground truth)
FN, false negative	The area of the exact area (ground truth), that is not a part of the TP area
FP, false positive	The area of the approximate (where-about) area that is not a part of the TP area

We randomly select a set of 100 queries and average the results. For each query, we randomly choose one moving object and a fixed tracking window size over the past *t* time units (e.g., the last 50 seconds). (We use the same tracking window size for all the queries that are grouped together.) For example, refer to Figure 6 for an example query with a tracking window of 4 time units.

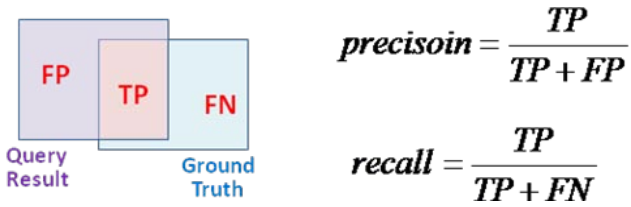

Fig. 5. Definitions of precision and recall for the performance study, defined in terms of *true positive* (TP), *false positive* (FP), and *false negative* (FN).

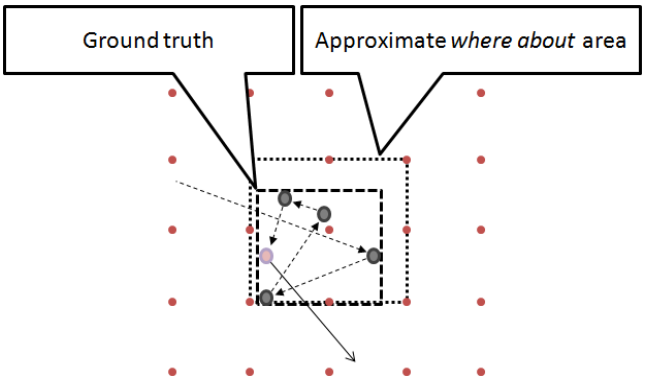

Fig. 6. An example true positive (darker dashes) shown overlaid on the where-about area returned from the query server (dotted rectangle) for a query that tracks the last 4 hops of the moving object.

5.1 Simulation Environment

The performance studies for this research have been conducted with the CAWA simulator, CAWASim. CAWASim is a discrete simulator employing the Random Waypoint model (RWP) to provide for the movement patterns of moving objects [30]. CAWASim simulates the injection of queries over randomly selected moving objects, of the query servers broadcasting and handing off queries to other query servers, mobile sensor to query server communication, the computation of where-about areas (the CAWA query result), and the calculation of ground truth and other metrics. CAWASim is implemented in C# on the .NET 3.5 architecture.

CAWASim simulates the interaction of the following objects:

- The query injector receives the query from the user and broadcasts it to all the query servers in the terrain.
- Query servers receive the queries from the query injector and maintain them in an internal list structure. Once the presence of the moving object is sensed by a mobile sensor, the query server has an idea of the approximate location of the moving object.

- Moving objects move through the terrain in accordance with the RWP model. Once the moving object reaches its destination it pauses for a randomly determined time, and then selects a new destination and speed and begins to move towards it; and on its way moves in and out of the sensing range of mobile sensors.
- Mobile sensors are either fixed at a particular location, or can move in accordance with the RWP model, depending upon the simulation settings. Each mobile sensor maintains a sensing window about it, which can sense the presence of all moving objects within its range. The mobile sensor only detects the presence, it cannot ascertain the precise coordinates of the moving object that is within its sensing range. For simplicity and to reduce calculation overhead, we model the sensing range of the moving object as a rectangle. We note that the rectangle can always be increased to contain any other sensing shape that may be more consistent with the broadcast pattern of an antenna.

Query servers receive updates from mobile sensors when the mobile sensor senses the presence of one or more moving objects within its detection window. If the query server receives data from multiple mobile sensors sensing the same moving object in an area where their query windows overlap, the query server can find the smaller area of intersection between all relevant mobile sensors sensing windows in order to compute a more precise approximate location.

Users pose queries to the query injector by specifying the ID of the target moving object that is to be tracked. The query server broadcasts the query to all of the other query servers, which are in communication with the query injector. Mobile sensors are equipped with GPS devices, and can sense the presence of moving objects that move within a threshold, or spatial sensing window, around each mobile sensor. However, a mobile sensor by itself cannot detect the exact location of a moving object. When a moving object is detected, the mobile sensor communicates its presence to its corresponding query server. The query server maintains a list of all the active queries it is currently monitoring, and once the presence of the moving object is reported by a mobile sensor, the query server has an approximate location for the moving object. It knows the moving object is bounded within the mobile sensor's sensing window, which is known to the query server. For example, if the mobile sensor has a 5 by 5 window in which it can sense moving objects, the query server knows the tracked moving object is somewhere within that area of 25 meters. However, if the mobile sensors are deployed in a way such that they have overlapping sensor areas, then when the query server finds that the moving object is detected by multiple mobile sensors, the query server can take the spatial intersection of the areas of their coverage, in order to deduce a more precise location for the moving object.

A fixed number of continuous queries are posed to the query server over a randomly selected set of moving objects. A selection of relevant configuration parameters are shown in Table 2.

Scenario 1: Dense coverage of mobile sensors

For the series of simulation runs presented here, we varied the size of the mobile sensor's window from 1 to 10 meters on a 1000 x 1000 meter grid, with each mobile

sensor (initially) positioned at each integer coordinate (that is, 1000000 mobile sensors arranged in the terrain in 1000 rows and 1000 columns). Two sets of results are presented: (1) where the mobile sensor locations are fixed (they do not move during the simulation) and (2) where they move in accordance with the RWP model with the same parameters as the moving objects (e.g., similar Pause Intervals, Speed Interval, etc.).

Table 2. Selected CAWASim configuration variables. A value of [*x*, *y*] means that a number is randomly chosen between *x* and *y* (inclusive) according to a uniform distribution.

Variable	Description	Example Value
RunCount	The number of simulation runs for a configuration	100
TerrainSize	Size of the simulated terrain in meters	1000x1000
NumMovingObjs	The number of moving objects simulated	500
MovObjSpeedIntvl	Movement speed of mobile objects selected from this interval	[1,100]
MovObjDestIntvl	Destination is randomly selected from the object's current position, in this range	[1,100]
PauseIntvl	Length of time a moving object pauses after its destination reached	[0,3] (seconds)
NumSensors	Number of mobile sensors in the simulation	100000
DetectionWindow	Size of the moving sensor's detection window	E.g., 10x10 represents a 10 meter by 10 meter window
DetectWinIntvl	Allows the detection window to be sized randomly (when the mobile sensor is created)	E.g., 0,20. Permits the default window to be expanded randomly
SensorsMove	Specifies if the mobile sensors move during the simulation	True/False

Table 3. Select simulation parameters common to all simulation runs

Variable	Value
RunCount	5 (the results from 5 simulation runs are averaged together)
NumMobileObjects	1000
PauseInterval	[1,5]

For this simulation the simulator was configured with values indicated in Table 4. The size of the query window for the mobile sensors was varied from 1 to 10 meters.

Table 5 presents simulation results where the sensors move during the simulation. Notice that good precision is achieved quickly without a significant amount of overlap in mobile sensor's sensing windows.

Table 4. Simulation parameters for scenario 1

Variable	Value
TerrainSize	1000x1000
SpeedInterval	[1,20]m/s: the speed is chosen (each time the moving object reaches its destination) uniformly from the range [1,20]
DestinationInterval	[1,20] meters
PauseInterval	[1,5] (node pauses between 1 and 5 seconds after reaching its destination)
QueryWindowSize	50 (the past 50 time units, or seconds, in the tracking window)

Table 5. Scenario 1 results – sensors mobile. The "size" column refers to the size (height and width) of the sensor's sensing area; a square in this case.

Size	Precision %	Recall %	Where-About Area	Ground Truth
1	72.32	100	3454.7	2642.6
2	96.11	100	2956.2	2866.4
3	95.45	100	2821.6	2735.1
4	95.85	100	2790.7	2703.2
5	95.90	100	2757.3	2670.7
6	95.51	100	2918.6	2829.9
7	95.62	100	3016.5	2911.9
8	95.71	100	3171.0	3067.8
9	95.57	100	3038.7	2940.0
10	96.33	100	3164.7	3072.1

Table 6. Scenario 1 results – sensors stationary

Size	Precision %	Recall %	Where-About Area	Ground Truth
1	75.08	100	2577.7	824.2624
2	95.36	100	2794.1	147.7275
3	95.98	100	2681.6	87.9034
4	96.71	100	3027.5	85.6379
5	95.70	100	2832.6	98.0112
6	95.88	100	2726.9	91.8981
7	95.42	100	2630.8	85.6811
8	96.28	100	3034.2	85.0350
9	95.67	100	2729.7	88.5538
10	95.89	100	3013.1	93.3657

The simulation results for Scenario 1 show that the precision and recall are very good and are robust to whether the mobile sensors are stationary or move during the simulation. Note that when the size of the mobile sensor's window is 1, that there is no overlap in the stationary scenario since the mobile sensors are positioned one unit apart from each other.

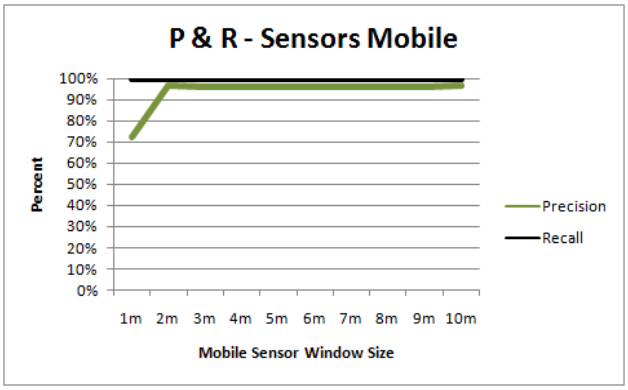

Fig. 7. Precision and Recall – with the mobile sensors moving in the terrain during the simulation, in accordance with the random waypoint model.

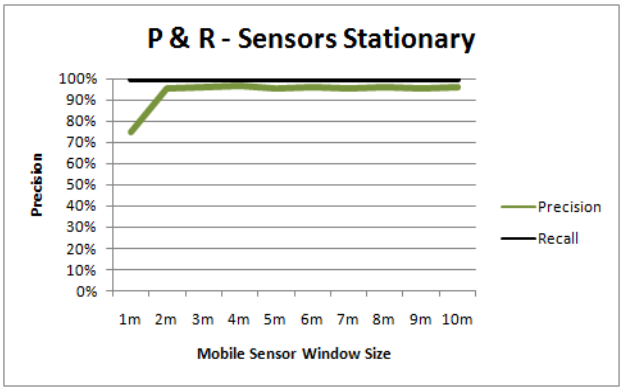

Fig. 8. Precision and Recall shown where the mobile sensors are stationary.

Table 7. Scenario 1 results – sensors mobile

Size	Precision %	Recall %	Where-About Area	Ground Truth
1	72.32	100	3454.7	2642.6
2	96.11	100	2956.2	2866.4
3	95.45	100	2821.6	2735.1
4	95.85	100	2790.7	2703.2
5	95.90	100	2757.3	2670.7
6	95.51	100	2918.6	2829.9
7	95.62	100	3016.5	2911.9
8	95.71	100	3171.0	3067.8
9	95.57	100	3038.7	2940.0
10	96.33	100	3164.7	3072.1

Scenario 2: Sparse Coverage of Mobile Sensors

For the series of simulation results presented here we varied the mobile sensor's window size from 1 to 10 meters on a 500 x 500 meter grid, with a mobile sensor node (initially) positioned in a grid of 250 rows by 250 columns. Two sets of results are presented: (1) where the mobile sensor locations are fixed (they do not move during the simulation) and (2) where they move in accordance with the RWP model with the same parameters as the moving objects (e.g., similar Pause Intervals, Speed Interval, etc.).

For this simulation the simulator was configured with values indicated in Table 8. The size of the query window for the mobile sensors was varied from 1 to 10 meters. *Coverage %* refers to the percentage of the terrain covered by mobile sensors sensing windows. For example, a coverage of 250% means that on the average each square is covered by 2.5 mobile sensors sensing windows.

Table 8. Select simulation parameters for scenario 2

Variable	Value
TerrainSize	500x500
SpeedInterval	[1,20]m/s: the speed is chosen
DestinationInterval	[1,20] meters
PauseInterval	[1,5] (node pauses between 1 and 5 seconds after reaching its destination)
QueryWindowSize	50 (the past 50 time units, or seconds, in the tracking window)

Table 9. Scenario 2 results – sensors stationary

Size	Precision %	Recall %	Where-About Area	Ground Truth	Coverage %
1	21.82	100	91747.2	12696.9	25
2	76.15	100	8924.4	7715.6	50
3	87.16	100	10954.0	10458.6	75
4	82.41	100	8970.2	8389.0	100
5	86.27	100	8800.4	8265.3	125
6	83.45	100	9019.1	8545.2	150
7	84.62	100	7762.7	7323.3	175
8	82.65	100	10337.8	9757.3	200
9	84.94	100	10645.9	10170.6	225
10	83.12	100	8881.9	8360.4	250

Table 10. Scenario 2 results – sensors mobile

Size	Precision %	Recall %	Where-About Area	Ground Truth	Coverage %
1	10.53	100	102957.5	10078.5	25
2	78.59	100	12025.4	10577.2	50
3	85.41	100	10650.1	9953.1	75
4	85.41	100	10650.1	9953.1	100
5	85.35	100	9985.0	9537.9	125
6	84.15	100	9291.7	8700.8	150
7	87.29	100	8786.2	8344.0	175
8	83.13	100	10474.0	9941.2	200
9	83.73	100	10025.9	9547.8	225
10	83.69	100	8096.6	7602.7	250

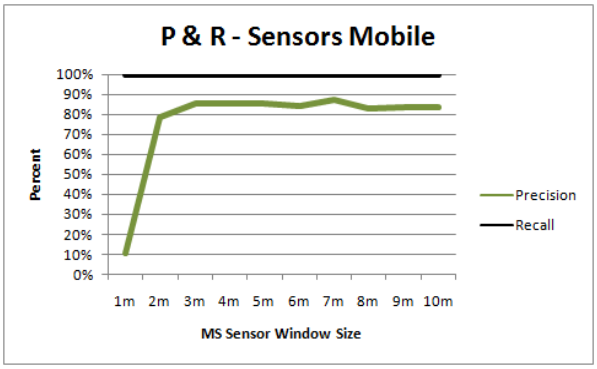

Fig. 9. Precision and Recall – sensors deployed in 250 rows and 250 columns in a 1000x1000 terrain, and the mobile sensors move in accordance with the RWP model.

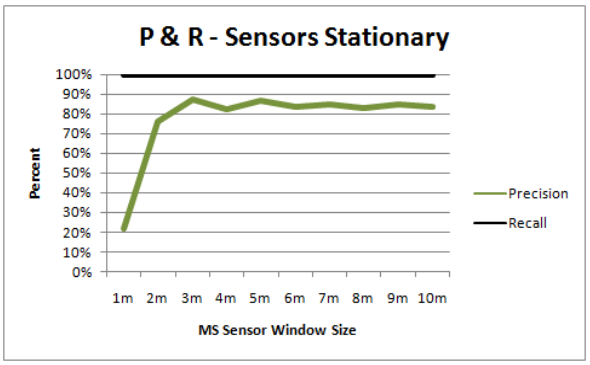

Fig. 10. Precision and Recall – stationary sensors, 250x250.

6 Conclusions

We present our research on Continuous Approximate Where-About queries, the first work, to our knowledge, on moving object databases that utilize mobile sensors that sense the presence of an object to deduct its approximate (where-about) location. Mobile sensors detect the presence of a moving object (not its exact location) and report presence updates to a query server, which knows the location and sensing area of each mobile sensor. When a moving object is detected by multiple mobile sensors, the query server takes the spatial intersection of the mobile sensors' sensing windows, as the approximate where-about area of the moving object. In many applications it is sufficient to know the approximate bounded location of a moving object, and in this research we show that even with a sparse population of mobile sensors the query server can deduce a fairly accurate approximation for the moving object, with a tracking window of t time units (e.g., the moving object's last t positions). Furthermore, we designed a system that is both cost effective and scalable.

To facilitate the evaluation of the proposed technique, we created the CAWASim, a discrete simulator that simulates moving objects and their interaction with queries, query servers and mobile sensors.

References

1. Ding S.: A survey on integrating MANETs with the Internet: Challenges and designs. Computer Communications, vol. 31, iss. 14 (September 2008)
2. Akyildiz I., Wang X. and Wang, W.: Wireless mesh networks: a survey. Computer Networks and ISDN Systems, vol. 47, iss 4, pp 445--487 (March, 2005)
3. Pfoser, D. and Jensen, C: Indexing of network constrained moving-object representations. Proc. of Advances of Spatial Databases, 6th Intl. Symp (SSD), pp 111--132 (1999)
4. Leonhardi, A. and Rothermel, K. A.: Comparison of Protocols for Updating Location Information. Cluster Computing, vol. 6, iss. 4. Kluwer Academic Publishers (2001)

5. Lee, K., Lee, Wang-Chen, and Zheng B.: Fast object search on road networks. Proc. of the Int'l Conf. on Extending Database Technology: Advances in Database Technology (EDBT'09) (2009)
6. Ortale, R., Ritacco, E., Pelekis, N., Trasarti, R., Cosa, G., Giannotti, F., Manco, G., Renso, C., and Theodoridis, Y.: The DAEDALUS framework: progressive querying and mining of movement data. Proc. of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems (2008)
7. Gedik, B., Wu, K.-L., and Liu, L.: Processing Moving Queries over Moving Objects Using Motion-Adaptive Indexes. IEEE Transactions on Knowledge and Data Engineering, vol. 18, iss 5. (May, 2006)
8. Chen, S., Jensen, C. and Lin, D.: A benchmark for evaluating moving object indexes. Proc. Of the VLDB Endowment. Publisher: VLDB Endowment, vol. 1, iss. 2 (2008)
9. Hernandez, C., Rodriguez, M. and Marin, M.: Complex Queries for Moving Object Databases in DHT-Based Systems. In Lecture Notes in Computer Science; Vol. 5168, pp. 424--433. Springer-Verlag, Berlin, Heidelberg (2008)
10. F. Brundick and G. Hartwig.: Model-based situational awareness. Proc. Of the Joint Service Combat Identification Systems Conf. (CISC'97) (1997)
11. Hightower, J. and Borrielo, G. Location systems for ubiquitous computing. IEEE Computer Magazine, 34, 8 (August, 2001)
12. Chamberlain, S.: Model-based battle command: A paradigm whose time has come. Proc. Of the Symposium on C2 Research and Technology, NDU (1995)
13. Want, R. and Schilit, B.: Expanding the horizons of location aware computing. IEEE Computer Magazine, vol. 34 iss. 8, pp. 31--34 (August, 2001)
14. Pitoura, E. and Samaras, G.: Locating objects in mobile computing. IEEE Transactions on Knowledge Data Engineering, vol. 13 iss. 4, pp. 571--592 (2001)
15. Abdessalem, T., Decreusefond, L., and Moreira, J.: Evaluation of Probabilistic Queries in Moving Objects Databases. In the Proc of International ACM Workshop on Data Engineering for Wireless and Mobile Access (MobiDE'06), Chicago, Illinois (2006)
16. Guting, R. and Schneider, M.: Moving Objects Databases. Morgan Kaufmann Publishers, (2005)
17. Cai, Y, Hua, K., Cao, G., and Xu, T.: Real-Time Processing of Range-Monitoring Queries in Heterogeneous Mobile Databases. IEEE Transactions on Mobile Computing, vol. 5 no. 7 (2006)
18. Hadjieleftheriou, M., Kollios, G., Tostras, V. and Gunopulos, D.: Indexing Spatiotemporal archives. VLDB Journal, vol. 15 iss.2 pp. 143--164 (2006)
19. Mokbel, M., Ghanem, T. and Aref, W.: Spatio-Temporal Access Methods. IEEE Data Engineering Bulletin, vol. 26 iss. 2, pp 40—49 (2003)
20. Pelanis, M., Saltenis, S. and Jensen, C.: Indexing the past, present, and anticipated future positions of moving objects. ACM Trans. on Database Systems, vol. 31 iss. 1, pp. 255—298 (2006)
21. Ni, J. and Ravishankar, C.: Indexing spatio-temporal trajectories with efficient polynomial approximations. IEEE Trans. on Knowledge Data Engineering, vol. 19 iss. 5, pp. 663--678 (2007)
22. Pfoser, D., Jensen, C. and Theodoridis, Y.: Novel Approaches to the Indexing of Moving Object Trajectories. Proc. of Very Large Databases (VLDB) (2006)
23. Pfoser, D. and Jensen, C.: Capturing the uncertainty of moving-object representations. Lecture Notes in Computer Science, vol. 1651, pp. 111--132 (1999)
24. Trajcevski, G., Wolfson, O., Zhang, F. and Chamberlain, S.: The geometry of uncertainty in moving object databases. Proc. of the ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE) (2003)

25. Sistla, A., Wolfson, O., Chamberlain, S. and Dao, S.: Querying the Uncertain Position of Moving Objects. In Temporal Databases: Research and Practice, vol. 1399, pp. 310—337 Springer-Verlag (1998)
26. Wolfson, O., Sistla, A., Xu, B., Zhou, J., Chamberlain, S., Yesha, Y. and Rishe, N.: Tracking moving objects using database technology in DOMINO. In Proc. of NGITS, vol. 1649 of Lecture Notes in Computer Science, pp 112—119 (1999)
27. Moreira, J., Ribeiro, C. and Abdessalem, T.: Query Operations for Moving Objects Database Systems. Proc. of ACMGIS, pp. 108--114 (2000)
28. Cheng, R., Prabhakar, K.-Y. and Liang, B.: An efficient location update mechanism for continuous queries over moving objects. In Information Systems, vol. 32, iss. 4, pp. 593--620. Elsevier Science, Ltd. (2007)
29. Abdessalem, T., Decreusefond, L., and Moreira, J.: Evaluation of Probabilistic Queries in Moving Objects Databases. Proc. of the ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE) (2006)
30. Bettsetter, C., Harenstein, H. and Perez-Costa, X.: Stochastic properties of the random waypoint mobility model. Wireless Networks, vol. 10, iss. 5. Kluwer Academic Publishers (2004)
31. Chen X., Schonfeld D., and Khokhar A.: Localization and Trajectory Estimation of Mobile Objects with a Single Sensor. Proc. of the IEEE/SP 14th Workshop on Statistical Signal Processing, vol. 0, pp 363--367 (2007)