Managing Uncertainty in Sensor Databases

Reynold Cheng Sunil Prabhakar
Department of Computer Science, Purdue University
Email: {ckcheng,sunil}@cs.purdue.edu

Abstract

Sensors are often employed to monitor continuously changing entities like locations of moving objects and temperature. The sensor readings are reported to a centralized database system, and are subsequently used to answer queries. Due to continuous changes in these values and limited resources (e.g., network bandwidth and battery power), the database may not be able to keep track of the actual values of the entities, and use the old values instead. Queries that use these old values may produce incorrect answers. However, if the degree of uncertainty between the actual data value and the database value is limited, one can place more confidence in the answers to the queries. In this paper, we present a framework that represents uncertainty of sensor data. Depending on the amount of uncertainty information given to the application, different levels of imprecision are presented in a query answer. We examine the situations when answer imprecision can be represented qualitatively and quantitatively. We propose a new kind of probabilistic queries called *Probabilistic Threshold Query*, which requires answers to have probabilities larger than a certain threshold value. We also study techniques for evaluating queries under different details of uncertainty, and investigate the tradeoff between data uncertainty, answer accuracy and computation costs.

1 Introduction

Sensors are often used to monitor the status of an environment continuously. The sensor readings are then reported to the application for making decisions and answering user queries. For example, a fire-alarm system in a building may employ temperature sensors to detect any abrupt change in temperature. An aircraft is equipped with sensors to track the wind speed, and radars are used to detect and report the aircraft's location to a military system. These applications usually include a database or server to which the sensor readings are sent. Limited network bandwidth and battery power imply that it is often not practical for the server to record the exact status of an entity it monitors at every time instant. In particular, if the value of an entity (e.g., temperature, location) being monitored is constantly evolving, the recorded data value may differ from the actual value. Querying the database can then produce *incorrect* results. Consider Figure 1(a), where sensors are used to monitor two moving objects a and b. If the location values recorded in the database (a_0 and b_0) are used, a will be the nearest neighbor of q. In reality, objects a and b have moved to positions a_1 and b_1, in which case b is the true nearest neighbor of q. In scenarios where critical decisions are made, producing incorrect answers may not be acceptable.

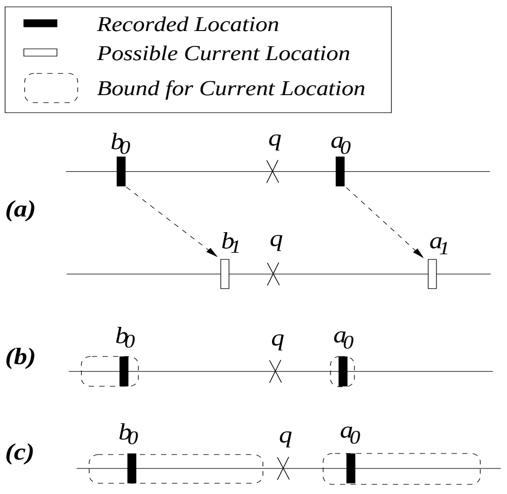

Figure 1: Example of data uncertainty and nearest-neighbor query.

Apparently, in a system that monitors constantly changing values, it is impossible to obtain meaningful query answers from the database. However, one can argue that the values of these objects usually cannot change drastically in a short period of time: the degree and/or rate of change of an object is constrained. Such information can help us alleviate the problem. In the previous example, if we can guarantee that at the time the query is evaluated, the actual locations of a and b are within certain bounds from a_0 and b_0 respectively (Figure 1(b)), then we can state with confidence that a is the nearest neighbor of q. In general, the objects' uncertainty may not allow us to identify a single object that is the nearest neighbor. For example, in Figure 1(c), *both* a and b can be the nearest neighbor of q since a may be farther than b from q, and vice versa. Therefore, if the state of an external entity is represented by an interval instead of an exact value, the answer can either be *certain* (Figure 1(b)) or *uncertain* (Figure 1(c)).

In this paper, we describe how the uncertainty of an entity can be modeled by an interval, and investigate how to answer a database query when intervals, instead of exact values, are given. We generalize the Future Temporal Logic (FTL) [11] for location queries, to queries with inherently uncertain sensor data as input. In particular, the `MUST` keyword is used in a query to specify that an answer must be certain to satisfy a query, while the `MAY` keyword allows uncertainty in answers. For example, in Figure 1(c), using the `MUST` keyword in the nearest neighbor query yields no answer, but a and b are returned when `MAY` is used instead.

While the `MAY` and `MUST` keywords help in querying inherently imprecise data, one may want to know further which object has a better chance of satisfying a query. For instance, in Figure 1(c), does a have a higher probability of being the nearest neighbor of q than b? In order to answer this question, not only does one need to have an interval to bound the uncertainty of an entity's value, he also needs to know the *probability distribution* of the value within the interval. With such additional information, one can augment different levels of confidence (e.g. as a probability) with each answer based on the uncertainty of the queried objects. This *probabilistic query* [13, 4, 3, 2] gives more useful information than the `MAY` and `MUST` keywords, since the user can know the likeliness of each answer in satisfying a query.

In this paper, we introduce the concept of a *probabilistic threshold query*. It is different from the probabilistic query in which one needs to specify the *minimum* probability value required for a query answer. As an example, suppose the probabilities of a and b for being q's nearest neighbor are 0.7 and 0.3 respectively. If the probability threshold is set to 0.5, then only a is the answer. Therefore, the probability threshold can be used to eliminate objects with small probability values. Compared with queries using `MAY` and `MUST`, it provides a more precise mechanism in handling uncertainty; compared with probabilistic queries, it allows a better control over the number of answers returned since a user may not be interested in answers that have very small chance of satisfying a query. We will also demonstrate how probabilistic threshold queries can be more efficient than probabilistic queries. To sum up, our contributions are:

- Propose a framework which is applicable to a wide range of constantly-evolving sensor data. The model encompasses different levels of understanding of data uncertainty, from no uncertainty to the most detailed uncertainty information;
- Extend the semantics of `MAY` and `MUST` keywords to support sensor data queries;
- Propose probabilistic threshold queries and illustrate why it is more efficient than probabilistic queries; and
- Study the relationship between data uncertainty, query complexity and answer imprecision.

The rest of this paper is organized as follows. In Section 2 we present a model that captures different details of data uncertainty. Section 3 discusses algorithms for evaluating queries over intervals, and Section 4 describes the evaluation of probabilistic threshold queries. In Section 5 we discuss some related work. Section 6 concludes the paper.

2 A Taxonomy of Data Uncertainty

We now present three forms of data uncertainty, namely *point uncertainty*, *interval uncertainty* and *probabilistic uncertainty*, whose differences are due to the amount of information known about a sensor data item.

1. Point Uncertainty The simplest data uncertainty model is to assume there is no uncertainty at all. Each data item is assumed to be a correct representation of the external entity being monitored. Specifically, let us assume that a real-valued attribute a of a set of database objects T is queried. [1] The ith object of T is named as T_i, and the value of a for T_i is called $T_i.a$ (where $i = 1, \ldots, |T|$). Database queries under the point uncertainty model simply uses the values of $T_i.a$'s as inputs. Although handling real values in a database query is easy, the query result is just an approximation of the true result. We have already illustrated that incorrect query results can be returned when a database attribute, $T_i.a$, is used to model a constantly changing quantity. We therefore need better data models to represent uncertainty.

2. Interval Uncertainty Different kinds of uncertainty models for constantly-evolving data have been proposed in the literature. One model assumes that the attribute value changes with known speed, but the speed may change each time the value is reported. Wolfson et al. [13] describe a model for locations of moving objects: at any point in time, the actual location is within a certain bound, d of its last reported location. If the actual location changes further than d, then the sensor reports its new location to the database and possibly changes d. Another model assumes that the object travels with known velocity along a straight line, but may deviate from this path by a certain distance [12].

Our second uncertainty model generalizes these models to support sensor-based applications. The basic idea is to use an interval (called *uncertainty interval*) to bound the possible values of $T_i.a$:

[1] Although we assume the domain of the data values is real, our model can be extended to higher dimensions.

Uncertainty Model	**Information Required**	**Query Type**
1. Point Uncertainty	$T_i.a$	Traditional queries
2. Interval Uncertainty	$T_i.a$, $U_i(t)$	Interval Queries with MAY and MUST keywords
3. Probabilistic Uncertainty	$T_i.a$, $U_i(t)$, $f_i(x,t)$	Probabilistic Queries, Probabilistic Threshold Queries

Table 1: Uncertainty models and query types

Definition 1: An **uncertainty interval** of $T_i.a$ at time instant t, denoted by $U_i(t)$, is an interval $[l_i(t), u_i(t)]$ such that $l_i(t)$ and $u_i(t)$ are real-valued functions of t, and that the conditions $u_i(t) \geq l_i(t)$ and $T_i.a \in [l_i(t), u_i(t)]$ always hold.

Therefore, in this model, all that is required is at the time of query execution the range of possible values of the attribute of interest is known. The uncertainty interval is often determined by the last recorded value ($T_i.a$), the time elapsed since its last update, and other application-specific assumptions. For example, $U_A(t)$ can contain all the values within a distance of $(t - t_{update}) \times v$ from its last reported value, where t_{update} is the time that the last update was obtained, and v is the maximum rate of change of the value.

3. Probabilistic Uncertainty This data model is proposed in [2]. Compared with the interval uncertainty, it requires one more piece of information – the probability density function or *pdf* of $T_i.a$ within $U_i(t)$. Assuming that $T_i.a$ is a continuous random variable, the formal definition of *uncertainty pdf* is given as follows:

Definition 2: An **uncertainty pdf** of $T_i.a$ at time t, denoted by $f_i(x,t)$, is a pdf of $T_i.a$, such that $f_i(x,t) = 0$ if $x \notin U_i(t)$.

Note that $f_i(x,t)$ must be a probability function bounded by $U_i(t)$ i.e., $f_i(x,y)$ equals 0 if (x,y) is outside $U_i(t)$. Also, in applications where the uncertainty pdf is not readily known, one may need to perform a statistical evaluation of the sensor data in order to choose a suitable pdf. An alternative is to simply assume that $T_i.a$ is uniformly distributed, i.e., $f_i(x,t) = 1/[u_i(t) - l_i(t)]$ for $T_i.a \in U_i(t)$. By employing a uniform distribution, one assumes the data value has an *equal* chance of locating anywhere in the uncertainty interval, which is a reasonable assumption. Because of its simple form, the uniform distribution can simplify query evaluation algorithms significantly.

Table 1 lists different types of data uncertainty and their relevant queries. When the knowledge about the data items increases, interval queries and probabilistic queries can be issued to provide more meaningful answers. We will examine these two classes of queries in the next sections. Unless stated otherwise, we assume queries are interested in the data at time instant t, the time at which the query is issued.

3 Interval Queries

As seen from Table 1, interval queries are those that treat data as uncertainty intervals. In this section, we present the semantics of interval queries. Then we illustrate how a range query is executed with uncertainty intervals as inputs.

3.1 The MAY and MUST keywords

Two important keywords, MAY and MUST, are used to control the behavior of an interval query. These two keywords follow the same semantics defined in [11] for moving objects.

When MUST is specified, the query engine can only return the answers that are certain to satisfy the query. For example, in Figure 1(b), Specifying MUST in the query will yield a as q's nearest neighbor, because the whole uncertainty interval of a is closer to q than any possible position of b. However, in Figure 1(c), neither a and b can be returned because we cannot tell for sure which one is the nearest neighbor of q.

When MAY is used, answers that have non-zero chance of satisfying the query are all returned. Hence a *may be* q's nearest neighbor in Figure 1(b); a and b *may be* q's nearest neighbor in Figure 1(c).

From these examples, it is clear that the set of answers returned by the MUST keyword is the subset returned by the MAY keyword. While a MUST query returns correct answers, its constraint may be so strict that none of the answer is returned. A MAY query, on the other hand, contains both correct and incorrect answers.

3.2 Evaluating an Interval Range Query

To illustrate how an actual interval query is evaluated, consider an Interval Range Query (IRQ), which is a range query for uncertainty intervals. Let $[l, u]$ be the query range specified by the user. When the MAY keyword is specified, IRQ returns all T_i's such that $T_i.a$ has a possibility of being inside $[l, u]$. Assume that $l_i(t) < u_i(t)$. Then we can evaluate IRQ by checking over each object in T to see if $U_i(t)$ overlaps $[l, u]$. An overlap indicates that $T_i.a$ may be inside $[l, u]$, so T_i is put into the result set, R. Figure 2 shows the complete algorithm.

```
1. $R \leftarrow \emptyset$
2. for $i \leftarrow 1$ to $|T|$ do
     (a) $OI \leftarrow U_i(t) \cap [l,u]$
     (b) if (width of $OI \neq 0$) then
           i. $R \leftarrow R \cup T_i$
3. return $R$
```

Figure 2: IRQ Algorithm Using the MAY Keyword.

When the MUST keyword is used, $U_i(t)$ has to be completely inside $[l,u]$ for T_i to be an answer. This is done by deleting Step 2(a), and replacing Step 2(b) by:

$$\textbf{if } (l \leq l_i(t) \textbf{ and } u_i(t) \leq u) \textbf{ then}$$

so that T_i is included in R when $T_i.a$ lies within $[l,u]$.

The range query algorithm in Figure 2 can actually be more efficient by using an interval tree that indexes one-dimensional intervals (see [7, 6] for details). However, these interval trees assume that the interval bounds are time-invariant, whereas the uncertainty bounds in our model are functions of time. Further studies are required to study how to use interval trees to index constantly changing bounds.

4 Probabilistic Threshold Queries

Interval queries using the MAY and MUST keywords offer a simple way to handle uncertainty. However, they only provide a qualitative treatment of data uncertainty. In particular, the MAY keyword does not specify *how likely* an answer can satisfy a query. Suppose two objects T_i and T_j appear in the results of an IRQ with the MAY keyword. The user may think that the two objects have the same chance of satisfying the query. In reality, the user is misled, since $T_i.a$ is completely inside $[l,u]$, while $U_j(t)$ only has a small overlap with $[l,u]$.

To solve this problem, we need to have a more quantitative treatment of uncertainty. We can replace the MAY and MUST keywords with a more specific requirement – a probability value – so that an object will only appear in the answer if its probability of satisfying the query is larger than that value. For example, one may ask "Which object has a probability of 0.7 or more in satisfying the range query?". Since any object with probability less than 0.7 is not part of the answer, the user can be more confident with the answers he gets.

An important question is: can we adopt the interval uncertainty data model to answer such queries? Unfortunately, the answer is not. Suppose an attribute with uncertainty interval $U_i(t)$ overlaps $[l,u]$. One is unable to tell the probability that $T_i.a$ is inside $[l,u]$, because no information about the distribution of $T_i.a$ in $U_i(t)$ is given. In general, we need to have an additional piece of information to answer such queries: the *pdf* of $T_i.a$ inside $U_i(t)$. This is where the probabilistic uncertainty data model comes into play. The probabilistic threshold query (PTQ) can now be defined as follows:

Definition 3: A **Probabilistic Threshold Query (PTQ)** with threshold p (where $0 < p \leq 1$) queries data under the probabilistic uncertainty model, and returns a set R of answers such that for each answer $r \in R$, r has a probability of p or more in satisfying the query.

Notice that an interval query with the MUST keyword is equivalent to a PTQ with $p = 1$, while the one with the MAY keyword can be viewed as a PTQ with p very close to 0. A PTQ allows a user to specify any $p \in (0,1]$ and thus provides more flexibility than an interval query.

4.1 Evaluating a Probabilistic Threshold Range Query

To understand how PTQ is evaluated, let us examine the Probabilistic Threshold Range Query (PTRQ). It is a range query that utilizes probabilistic uncertainty and returns T_i's such that the probability that $T_i.a$ is inside $[l,u]$, denoted by p_i, is greater than or equal to p. Before presenting its evaluation algorithm, it is useful to review the Probabilistic Range Query (PRQ) discussed in [2]. PRQ is different from PTRQ because PRQ imposes no probabilistic constraint on the result i.e., it returns all tuples (T_i, p_i) with all $p_i > 0$. Also, PRQ returns p_i to the user, but PTRQ does not.

```
1. $R \leftarrow \emptyset$
2. for $i \leftarrow 1$ to $|T|$ do
     (a) $OI \leftarrow U_i(t) \cap [l,u]$
     (b) if (width of $OI \neq 0$) then
           i. $p_i \leftarrow \int_{OI} f_i(x,t)dx$
           ii. if $p_i \neq 0$ then $R \leftarrow R \cup (T_i, p_i)$
3. return $R$
```

Figure 3: PRQ Algorithm.

Figure 3 shows how PRQ is evaluated. Similar to IRQ, the overlapping interval OI of each $U_i(t)$ and $[l,u]$ is evaluated (Step 2a). Next, p_i is computed by integrating $f_i(x,t)$ over OI (Step 2b(i)). If $p_i \neq 0$, (T_i, p_i) is inserted into set R that stores the result (Step 2b(ii)).

To evaluate PTRQ, we can simply change Step 2b(ii) to

$$\textbf{if } p_i \geq p \textbf{ then } R \leftarrow R \cup T_i$$

so that the result only contains T_i's with p_i's greater than or equal to p. Unfortunately, this simple approach is not very efficient. This is because we need to evaluate an integral

Query Type	Uncertainty Constraint	Example	Answer	Complexity
1. Traditional query	None	Range Query	May be incorrect	Low
2. Interval query	`MAY` and `MUST`	IRQ	Imprecise (qualitative)	Medium
3. Probabilistic threshold query	Probabilistic threshold	PTRQ	Imprecise (probabilistic)	High

Table 2: Query properties under different uncertainty models.

for each T_i in Step 2b(i) numerically, which involves dividing the area under the curve of $f_i(x,t)$ into small equal-sized stripes and summing up the area of each stripes. [2] We can reduce the computation cost by exploiting the probabilistic threshold p:

1. Estimate the upper bound of the integral before evaluating it. Specifically, divide OI into a few sub-intervals, where the height of each sub-interval is equal to the largest value of $f_i(x,t)$ in that sub-interval. If the sum of the areas of these large sub-intervals is smaller than p, we can immediately conclude that $p_i < p$ and T_i cannot be the answer.

2. To evaluate the integral, the area of each stripe is evaluated and added to an intermediate sum p_{inter}, where integration is completed when p_{inter} equals to p_i. Once $p_{inter} \geq p$, we can deduce that $p_i \geq p$ and stop computing the integral.

3. If the interval outside $[l,u]$ (i.e., $U_i(t) - OI$) has a much shorter length than OI, we can evaluate $\int_{U_i(t)-OI} f_i(x,t)dx$ instead. If the value of this integral is smaller than p, then p_i must be larger than p.

From the above discussions, we observe that by imposing a probabilistic threshold to a probabilistic query, various optimization techniques can be employed to improve its efficiency. The techniques for exploiting the threshold depend on the class of the query, since each query class has its own evaluation algorithm (see [2] for a classification of probabilistic queries). An interesting future work is to explore the efficient evaluation of PTQ for different query classes.

4.2 A Comparison of Imprecise Queries

Now let us compare traditional queries, interval queries and probabilistic threshold queries. A traditional query does not consider any data uncertainty information and is thus the most efficient. However, its query answer can be completely incorrect. An interval query handles interval uncertainty, and is thus more complicated than traditional queries. The answers it provides, however, are more meaningful than those provided by traditional queries, where the imprecision of answers is expressed through the use of `MAY` and `MUST` keywords. Finally, a probabilistic threshold query runs under the probabilistic uncertainty model. Since the *pdf* of the attribute value is considered, its evaluation algorithm is costly (as illustrated by the difference in evaluating an IRQ and PTRQ). Nevertheless, it treats imprecise answers quantitatively with probabilistic guarantees, so that a user can have a better understanding of the answers. Table 2 summarizes the properties, answer precision and computational complexity of the three query types.

[2] In general we need to evaluate the integral numerically, unless $f_i(x,t)$ is a simple function like the uniform distribution.

5 Related Work

The constantly changing sensor data studied in this paper are collectively known as *dynamic attributes* [11]. Their main property is that their values change over time even if they are not explicitly updated in the database. In [13], Wolfson et al. present a moving object model where each moving object is equipped with a facility to detect the deviation of its actual location from the location value in the DBMS. A threshold value d is defined in such a way that if the deviation is larger than it, then an update of the location of that object is sent to the DBMS. If an object moves without any route, the uncertainty is a circle with radius d bounding the location of the object. This model can be generalized to the interval uncertainty model, which bounds the uncertainty of any continuously changing sensor data. The probabilistic uncertainty model is discussed in [4] for moving objects and in [2] for continuously sensor data. The essence of these models is captured by our more general and flexible framework. A system designer can choose how detailed data uncertainty should be represented, depending on system resources and user requirements.

A spatio-temporal query language, called the Future Temporal Logic (FTL), has been proposed in [11] for querying moving object locations. The authors define the “may” and “must” semantics for FTL: the former semantic specified that the answer to a query has a probability of being incorrect, while the latter one requires the answer to be correct. Another similar work in spatio-temporal query language is by Abdessalem et al. [1]. We apply their concepts to interval queries for inherently uncertain sensor data.

There are a number of works about evaluation of intervals.

Olston et al. discuss the problem of balancing the trade-off between precision and performance for querying replicated data [9, 8, 10]. In their model, the cache in the server cannot keep track of the exact values of sensor sources due to limited network bandwidth. Instead of storing the actual value in the server's cache, an interval for each item is stored, within which the current value must be located. A query is then answered by using these intervals, together with the actual values fetched from the sources. The problem of minimizing the update cost within an error bound specified by aggregate queries is studied. Khanna et al. [5] extend Olston's work by proposing an online algorithm to identify a set of elements with minimum update cost so that a query can be answered within an error bound. Manolopoulos et al. [7] discuss an efficient interval tree to facilitate the execution of intersection queries over intervals, and Kriegel et al. [6] propose an implementation of interval trees on top of relational tables and database queries.

For probabilistic queries, Wolfson et al. [13] discuss range queries for moving objects, while Cheng et al. [4, 3] describe nearest-neighbor query algorithms under different object movement models. A general classification, evaluation and quality of different types of probabilistic queries for sensor data are presented in [2]. Here we propose the probabilistic threshold query, which can be more efficient than probabilistic queries (recall the discussions of PTRQ in Section 4.1). It also allows users to have more control over the answers, where only those with probabilities larger than the threshold are returned.

6 Conclusions

Uncertainty is inherent with constantly-changing sensor data. If a database system makes use of the sensor data without being aware of its uncertainty, a query can yield incorrect results. We tackle this problem by introducing a data uncertainty framework that possesses different details of uncertainty information. By using uncertainty information wisely, queries can produce more meaningful results.

Depending on the amount of uncertainty given, queries can produce different forms of imprecise results. Through the example of a simple range query, we illustrate how an interval query handles uncertainty interval qualitatively through the use of `MAY` and `MUST` keywords, and how a PTQ produces results with probabilistic guarantees. While the result of a PTQ is the most informative, it is also the most expensive. However, a PTQ can be more efficient than a probabilistic query if the probabilistic threshold constraint is exploited. Our future work is to examine how to make other queries, such as nearest-neighbor queries, to execute faster through the use of probabilistic threshold information.

References

[1] T. Abdessalem, J. Moreira, and C. Ribeiro. Movement query operations for spatio-temporel databases. In *Proc. 17èmes Journées Bases de Données Avancées (BDA'01)*, Agadir, Maroc, October 2001.

[2] R. Cheng, D. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, June 2003.

[3] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. In *IEEE Transactions On Knowledge and Data Engineering (to appear)*, 2004.

[4] R. Cheng, S. Prabhakar, and D. V. Kalashnikov. Querying imprecise data in moving object environments. In *Proc.of the Intl Conf. on Data Engineering (ICDE'03)*, 2003.

[5] S. Khanna and W.C. Tan. On computing functions with uncertainty. In *20th ACM Symposium on Principles of Database Systems*, 2001.

[6] H. Kriegel, M. Potke, and T. Seidl. Managing intervals efficiently in object-relational databases. In *Proc. of the 26th Intl. Conf. on VLDB*, Cairo, Egypt, 2000.

[7] Y. Manolopoulos, Y. Theodoridis, and V.J. Tsotras. Chapter 4: Access methods for intervals. In *Advanced Database Indexing*. Kluwer, 2000.

[8] C. Olston, Boon Thau Loo, and J. Widom. Adaptive precision setting for cached approximate values. In *Proc. of the ACM SIGMOD 2001 Intl. Conf. on Management of Data*, 2001.

[9] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *Proc. of the 26th Intl. Conf. on Very Large Data Bases*, 2000.

[10] C. Olston and J. Widom. Best-effort cache synchronization with source cooperation. In *Proc. of the ACM SIGMOD 2002 Intl. Conf. on Management of Data*, pages 73–84, 2002.

[11] P. A. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Querying the uncertain position of moving objects. In *Temporal Databases: Research and Practice*. 1998.

[12] Goce Trajcevski, Ouri Wolfson, Fengli Zhang, and Sam Chamberlain. The geometry of uncertainty in moving object databases. In *EDBT, 8th International Conference on Extending Database Technology*, pages 233–250. Springer, March 2002.

[13] O. Wolfson, P. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, 7(3), 1999.