Orion 2.0: Native Support for Uncertain Data

Sarvjeet Singh, Chris Mayfield, Sagar Mittal,
Sunil Prabhakar, Susanne Hambrusch, Rahul Shah*
Department of Computer Science, Purdue University
West Lafayette, Indiana, USA
{sarvjeet, cmayfiel, smittal, sunil, seh, rahul}@cs.purdue.edu

ABSTRACT

Orion is a state-of-the-art uncertain database management system with built-in support for probabilistic data as first class data types. In contrast to other uncertain databases, Orion supports both attribute and tuple uncertainty with arbitrary correlations. This enables the database engine to handle both discrete and continuous pdfs in a natural and accurate manner. The underlying model is closed under the basic relational operators and is consistent with Possible Worlds Semantics. We demonstrate how Orion simplifies the design and enhances the capabilities of two example applications: managing sensor data (continuous uncertainty) and inferring missing values (discrete uncertainty).

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*; H.2.8 [**Database Management**]: Applications

General Terms

Design, Management, Theory

1. INTRODUCTION

Probabilistic and uncertain data management have recently received much attention in the database community (see [7] for related work). Uncertainty is prevalent in numerous application domains, ranging from information extraction and integration to scientific data management and sensor databases. Orion[1] is a general-purpose uncertain DBMS that unifies the modeling of probabilistic data across applications. This in turn provides additional opportunities to the query engine for indexing and optimization.

One motivating example is a data cleaning system that automatically detects and corrects errors. Since conventional database management systems assume data to be certain and precise, the software must either construct its own probabilistic model for the data, or simply pick one of the alternative values to store in the underlying database. This leads to a no-win situation: the first option significantly complicates the queries, while the second technique results in a substantial loss of information.

*Work done while at Purdue University. Current affiliation: Louisiana State University, Baton Rouge, Louisiana, USA.
[1]See `http://orion.cs.purdue.edu/`

SIGMOD'08, June 9–12, 2008, Vancouver, BC, Canada.
ACM 978-1-60558-102-6/08/06.

The Orion system provides a better solution: built-in support for uncertainty at the database level. By extending the query processing engine of PostgreSQL, Orion natively manages uncertain data modeled as arbitrary joint probability distributions. "Orion 2.0" is a complete redesign and rewrite of its predecessor "U-DBMS" [1], and includes the following new and innovative contributions:

- An integrated implementation (*within* PostgreSQL) of the "PDF Attributes" data model, which is consistent with *Possible Worlds Semantics* (PWS) and supports both continuous and discrete uncertainty [7].
- Efficient access methods for querying uncertain data, including three index structures based on R-trees, signature trees, and inverted indexes [3, 5].
- Improved query optimization, join algorithms, and selectivity estimation by gathering and exploiting additional statistics over probabilistic data types [2, 6].
- Integration with PL/R for graphical visualization of and statistical inference over uncertain data [4].

The fundamental difference between Orion and related projects is indeed its support of attribute-level continuous uncertainty, which enables the system to represent probabilistic data in a natural and efficient manner.

2. THE ORION 2.0 DATA MODEL

Orion 2.0 has two major kinds of attributes – *uncertain* (or pdf attributes) and *certain* (or precise attributes). A database table T is defined by a *probabilistic schema* (Σ_T, Δ_T) consisting of a *schema* (Σ_T) and *dependency information* (Δ_T). The schema Σ_T is similar to the regular relational schema and specifies the names and data types of the table attributes (both certain and uncertain). The dependency information Δ_T identifies the attributes in T that are jointly distributed (i.e. correlated). For each dependent set of attributes in T, the model maintains a *history* Λ.

PDF attributes: The uncertainty in many applications can be expressed using standard distributions. Orion has built-in support for commonly used continuous (e.g. Gaussian, Uniform) and discrete (e.g. Binomial, Poisson) distributions. These uncertain values are processed symbolically in the database. When the underlying data cannot be represented using standard distributions, Orion automatically converts them to approximate distributions, including histograms and discrete sampling.

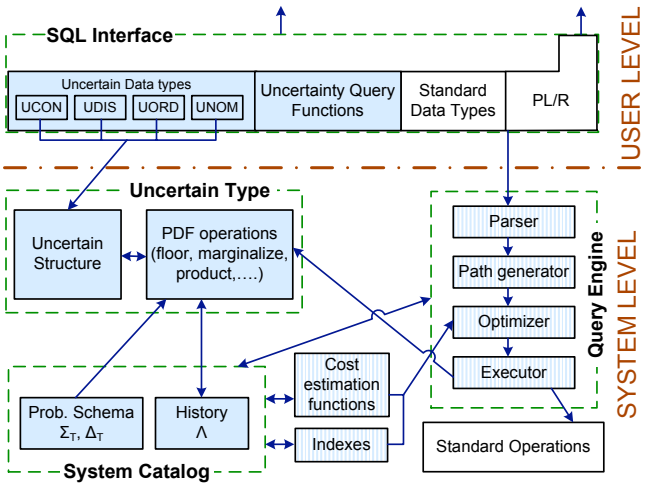

Figure 1: Orion 2.0 Architecture

Correlations and missing data: Correlated attributes in a table T (given by Δ_T) are represented by a single joint distribution. An important feature of Orion is its support for *partial pdfs*. A partial pdf is a distribution that sums (or integrates) to less than 1. This feature allows us to represent *missing tuples*. If the joint pdf of a tuple (obtained by multiplying the individual pdfs of attributes) sums up to x, then $1-x$ is the probability that the tuple does not exist in the database.

Historical dependencies: In addition to the pdfs, for each dependent group of uncertain attributes present in Δ_T, we store its *history* Λ. While dependency information (or joint pdfs) express *intra-tuple* dependencies at the schema level, history captures the *inter-tuple* dependencies at the instance level. The history of a given set of uncertain attributes denotes the attribute sets from which it is derived, and is used while performing operations on the tuples to ensure that the results are consistent with PWS.

PDF operations: Correct evaluation of select-project-join queries under PWS reduces to three fundamental operations on pdfs: *floor*, *marginalize*, and *product* (see [7] for a detailed discussion). These operations use the information maintained by dependency sets in Δ_T and histories in Λ to detect any correlations and handle them appropriately. The standard relational operations remain unchanged for the certain attributes in the database.

3. SYSTEM IMPLEMENTATION

Orion is primarily written in C, with some portions at the user level in PL/pgSQL. Figure 1 gives a high level overview of the system architecture. The shaded regions represent new components that correspond to the primary features of the Orion data model. Partially shaded boxes highlight portions of the PostgreSQL backend we extended to support queries over uncertain data. Clear boxes (which include the majority of the PostgreSQL backend) indicate components we have not modified.

Query interface: One underlying goal in the design of Orion was to support uncertain data with minimal changes to SQL. The resulting user interface is standard SQL plus a handful of data types and built-in functions for manipulating probabilistic data. These include, for example, evaluating the cdf of an uncertain attribute, and converting symbolic pdfs into approximations. In addition, we have integrated our system with PL/R [4], an extension to PostgreSQL that allows the user to write SQL statements and functions in the R programming language. "R is a free software environment for statistical computing and graphics,"[2] and provides elegant visualization of uncertain pdfs in the Orion client.

Uncertain data types: Orion supports four main types of uncertain data attributes:

1. Continuous Numeric (`ucon`) – Each data item has an associated probability density function for evaluating the probability of any given value.
   Example: Temperature or voltage from a sensor.
2. Discrete Numeric (`udis`) – Each data item has a probability distribution function, which stores the frequencies of the alternative values.
   Example: Number of neighbors in a mobile network.
3. Ordered Categorical (`uord`) – Similar to discrete numeric, each data type comes with a pdf that stores probabilities for each category.
   Example: Fuzzy data value, e.g. low or high.
4. Unordered Categorical (`unom`) – Same as above, except there is no logical ordering between categories.
   Example: Document classification or generic type.

Internal representation: All uncertain attributes are stored internally using a data structure named `Uncertain`. This type is hidden from the user, and is only accessible through the four SQL data types listed above. Consequently, the data structure is generic and represents all possible types of uncertainty pdfs. In particular, it can represent both independent and joint distributions. When multiple attributes are correlated, the system automatically stores the number of dimensions, the type of each dimension, and the resulting joint pdf in a single data instance.

In addition to the pdf, `Uncertain` also maintains a list of floored regions and historical dependencies that are due to operations on pdfs. Probabilistic schemas (i.e. dependency sets) for each table are stored in the system catalog. All of this information is used by internal functions to detect correlations while performing pdf operations. The demo highlights the relative trade-offs of the two different options for storing histories. These are either: 1) to store the top-level *ancestors*, or 2) to store the immediate *parents* of the attribute sets in a graph-like structure.

Indexes and query optimization: The standard cost estimation and indexing techniques built into PostgreSQL are not appropriate for uncertain data. Orion provides novel query cost estimation techniques that are used for optimizing the generated query plans involving uncertain data [6]. In addition to cost estimation, Orion also includes a number of uncertainty indexing methods and join algorithms for efficient execution of specialized queries [2, 3, 5].

One major advantage with the design and implementation of Orion is that there is virtually no system overhead in the absence of uncertain data. The modifications for uncertain data support are for the most part self-contained, and operate side by side with the standard indexing and query optimization components.

[2]See `http://www.r-project.org/`

```
CREATE TABLE location (
  id integer, ts time,
  xloc ucon, yloc ucon, room udis, -- unc. types
  PRIMARY KEY (id, ts),
  DEPENDENT (xloc, yloc) );      -- prob. schema

INSERT INTO location VALUES (
  1, '2008-06-09 14:05:27',
  'prod( norm(5,3) , norm(7,3) )',  -- 2D pdf
  'dist( 2 : 0.75 , 3 : 0.25 )' );  -- 1D pdf

SELECT * FROM location          -- floored pdfs
  WHERE xloc > 5 and yloc < 5;  -- with history
```

Figure 2: Example SQL Queries

4. DEMONSTRATION CONTENT

We demonstrate the benefit of processing uncertainty at the database level by running two example applications interactively. Orion seamlessly manages all types of uncertainty, whether inherent in the base data or generated via probabilistic operations over standard data.

4.1 Example Applications and Datasets

The first application monitors the movement of people within a building using 802.11-based sensors that report approximate locations in real-time. Using a dataset collected during training exercises at the Purdue Homeland Security Institute, we show how Orion effectively stores and indexes uncertain position measurements as gaussians derived from the sensor specifications. We also highlight how accurate selectivity estimation over such data guides the query optimizer in when to use the uncertain index structures. The presence of these features increases the overall accuracy.

The second application is a genealogical database that leverages discrete uncertainty to infer missing information. We present a basic technique for estimating unknown birth and death years for individuals based on their relatives. Starting with known events and domain knowledge, our system iteratively propagates estimated ranges from parents to children and vice versa. Orion tracks the change in uncertainty during each iteration, and automatically handles correlations when combining multiple values in the base data.

4.2 Outline of the Demonstration

Creating uncertain data: We first introduce the Orion data model through a series of SQL statements that create and populate uncertain data tables for the aforementioned applications. These examples illustrate both inter-tuple and intra-tuple dependencies, and we show how the system efficiently tracks and accurately reports probability values.

Probabilistic queries: Next, we demonstrate the two example applications from the user's perspective, and display a real-time SQL trace on the database server. We also show several non-trivial queries over uncertain data and their resulting execution plans.

Optimization and indexes: To highlight the performance features of Orion, we demonstrate common queries with and without indexes. We also turn off cost estimation and show how the inappropriate use of indexes would have affected efficiency.

Visualization of results: Using PL/R, we show several interesting plots of the uncertain data constructed in the previous steps. We also use a custom Orion client to illustrate historical dependencies.

Behind the scenes: Since Orion is built inside of the database engine, we conclude by demonstrating the low-level features of our system using standard PostgreSQL tools. These include psql (the command line client) and pgAdmin III (a graphical administration platform).

5. FUTURE DIRECTIONS

In this demonstration we have highlighted only two of the numerous applications that will immediately benefit from advancements in probabilistic data management. The Orion project aims to build a general-purpose uncertain DBMS to support both current and forthcoming applications. We are exploring several avenues of future work which are essential to providing a comprehensive database solution:

- Probabilistic extensions to (and semantics for) other standard database features including aggregation, duplicate elimination, and key constraints.
- Approximation methods for calculating probabilities in result sets with complex historical dependencies.
- Efficient processing of specialized queries, including k-nearest neighbor queries over uncertain data.
- If necessary, additional SQL language extensions.

Research and development of a database system that supports uncertain data will advance scientific understanding and enable future work in a variety of fields. But whether emerging applications use databases simply as an information storage technology rather than an effective data management solution depends on to what extent they can reason about and make use of the uncertainty of data directly.

6. REFERENCES

[1] R. Cheng, S. Singh, and S. Prabhakar. U-DBMS: A Database System for Managing Constantly-Evolving Data. In *31st Intl. Conference on Very Large Data Bases*, 2005.

[2] R. Cheng, S. Singh, S. Prabhakar, R. Shah, J. Vitter, and Y. Xia. Efficient Join Processing over Uncertain Data. In *ACM 15th Conference on Information and Knowledge Management*, 2006.

[3] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter. Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data. In *30th Intl. Conference on Very Large Data Bases*, 2004.

[4] J. E. Conway. PL/R - R Procedural Language for PostgreSQL. `http://www.joeconway.com/plr/`, 2008.

[5] S. Singh, C. Mayfield, S. Prabhakar, R. Shah, and S. Hambrusch. Indexing Uncertain Categorical Data. In *IEEE 23rd Intl. Conference on Data Engineering*, 2006.

[6] S. Singh, C. Mayfield, R. Shah, S. Prabhakar, and S. Hambrusch. Query Selectivity Estimation for Uncertain Data. In *20th Intl. Conf. on Scientific and Statistical Database Management*, 2008.

[7] S. Singh, C. Mayfield, R. Shah, S. Prabhakar, S. Hambrusch, J. Neville, and R. Cheng. Database Support for Probabilistic Attributes and Tuples. In *IEEE 24th Intl. Conference on Data Engineering*, 2008.