ProbDB: Efficient Execution of Aggregate Queries over Probabilistic Data

Guillaume Verger, Reza Akbarinia, Patrick Valduriez

ProbDB: Efficient Execution of Aggregate Queries over Probabilistic Data

Guillaume Verger[1], Reza Akbarinia[2], Patrick Valduriez[2],

INRIA and LIRMM, Montpellier, France

[1]Verger@lirmm.fr, [2]FirstName.LastName@inria.fr

Abstract

Les requêtes d'agrégats sont essentielles dans de nombreuses applications qui gèrent des données incertaines. Cependant, calculer de manière efficace ces requêtes est généralement difficile. Dans cette démonstration nous présentons ProbDB, un système de gestion de bases de données probabilistes. Nous montrons en particulier l'efficacité de ProbDB lors du calcul de requêtes avec agrégats tels que SUM et COUNT. L'application sur laquelle notre démonstration se repose est une application médicale, et elle montre la rapidité avec laquelle ProbDB répond aux requêtes d'agrégats probabilistes.

1. Introduction

In the context of probabilistic databases [1], aggregate (or *aggr* for short) queries, in particular SUM and COUNT queries, are crucial for many applications that need to deal with uncertain data. Let us give a motivating example from the medical applications domain.

Example 1: Remote health monitoring. Consider a medical center that monitors key biological parameters of remote patients at their homes, *e.g.* using sensors in their bodies. The sensors periodically send to the center the patients' health data, *e.g.* blood pressure, hydration levels, thermal signals, etc. For high availability, there are two or more sensors for each biological parameter. However, the data sent by sensors may be uncertain, and the sensors that monitor the same parameter may send inconsistent values. There are approaches to estimate a confidence value for the data sent by each sensor, *e.g.* based on their precision [1]. According to the data sent by the sensors, the medical application computes the number of required human resources, e.g. nurses, and equipments for each patient. One important query in this application is "return the sum of required nurses".

We are interested in the queries that return all possible aggregate values and their probabilities. This kind of query, which we call ALL_AGR (also known as aggregate probability distribution) is very important for our underlying applications, e.g. for the above medical applications. For example by using ALL_ASUM (all possible sum results and their probabilities) we can generate the aggregate probability distribution of attribute summation which is needed for decision makings, e.g. how many nurses are needed to cover all emergency patients with a probability higher than 99%.

A naïve algorithm for evaluating ALL_AGR queries is to enumerate all possible worlds, i.e. all possible database instances, compute sum in each world, and return the possible sum values and their aggregated probability. However, this algorithm is exponential in the number of uncertain tuples.

In this demonstration, we present ProbDB, a probabilistic database system for managing probabilistic data. In particular, we show the efficiency of ProbDB for processing aggregate queries such as SUM and COUNT. Our demonstration application is the remote health monitoring application described in Example 1. We run our prototype over a predefined dataset (which can be modified online) of this application, and show how fast it answers to

aggregate queries. The user can modify the queries and data to visually see the effect on the probability distribution function.

The rest of the paper is organized as follows. In Section 2, we present some technical basis of our prototype, e.g. the probabilistic data models and some intuitions about our algorithms. In Section 3, we describe our ProbDB prototype. In Section 4 we present the application scenarios that will be shown to users during our demonstration. Section 5 concludes.

2. Technical Basis

In this section, we first introduce the two probabilistic data models that ProbDB supports. Then, we give the intuition about our SUM and COUNT query processing in ProbDB (for more details see [2]).

2.1 Supported Probabilistic Models

In ProbDB, two probabilistic data models, which are frequently used in probabilistic databases, are supported: tuple-level and attribute-level models. They are defined as follows.

Tuple-level model. In this model, each uncertain table T has an attribute that indicates the *membership probability* (also called existence probability) of each tuple in T, *i.e.* the probability that the tuple appears in a possible world. In this paper, the membership probability of a tuple t_i is denoted by $p(t_i)$.

Attribute-level model. In this model, each tuple t_i has at least one uncertain attribute, e.g. α, and the value of α in t_i is chosen by a random variable X. Depending on the probability density function (pdf) of X, the values of α in t_i may be m values $v_{i,1}, \ldots, v_{i,m}$ with probabilities $p_{i,1}, \ldots, p_{i,m}$ respectively Note that for each tuple we may have a different pdf.

2.2 Algorithmic Basis

In ProbDB, almost all aggregate functions are processed using recursive algorithms. Below, we give the intuition about the ALL_SUM algorithm that is in charge of executing SUM queries. It is also used for answering COUNT queries. For more details, the reader is referred to [2].

Let $t_1, \ldots, t_n$ be the tuples of the given database which is under the tuple-level model. Let DB^j be a database involving the tuples $t_1, \ldots, t_j$, and W^j be the set of all possible worlds in DB^j. Let $ps(i, j)$ be the probability of having $sum = i$ in DB^j. We develop a recursive approach for computing $ps(i, j)$.

2.2.1 Base

Let us consider DB^1, *i.e.* the database that involves only the tuple t_1. Let $p(t_1)$ be the membership probability of t_1, and $val(t_1)$ be the aggr value of t_1. In DB^1, there are two possible worlds: 1) $w_1=\{\}$, in which t_1 does not exist, so its probability is $(1- p(t_1))$; 2) $w_2=\{t_1\}$, in which t_1 exists, so the probability is $p(t_1)$. In w_1, we have $sum=0$, and in w_2 we have $sum=val(t_1)$. If $val(t_1) = 0$, then always we have $sum=0$ because in both w_1 and w_2 sum is zero.

2.2.2 Recursion Step

Now consider DB^{n-1}, *i.e.* a database involving the tuples $t_1, \ldots, t_{n-1}$. Let W^{n-1} be the set of possible worlds for DB^{n-1}, i.e. set of possible instances for DB^{n-1}. Let $ps(i, n-1)$ be the probability of having $sum=i$ in DB^{n-1}, *i.e.* the aggregated probability of the DB^{n-1} worlds in which we have $sum=i$. Now, we construct DB^n by adding t_n to DB^{n-1}. Notice that the set of

DB^n possible worlds, denoted by W^n, are constructed by adding or not adding the tuple t_n to each world of W^{n-1}. Thus, in W^n, there are two types of worlds: 1) the worlds that do not contain t_n, denoted as W^n_1; 2) the worlds that contain t_n, denoted as W^n_2.

For each world $w \in W^n_1$, we have the same world in DB^{n-1}, say w'. Let $p(w)$ and $p(w')$ be the probability of worlds w and w'. The probability of w, i.e. $p(w)$, is equal to $p(w') \times (1 - p(t_n))$, because t_n does not exist in w even though it is involved in the database. Thus, in W^n_1 the sum values are the same as in DB^{n-1}, but the probability of $sum=i$ in W^n_1 is equal to the probability of having $sum=i$ in DB^{n-1} multiplied by the probability of non-existence of t_n. In other words, we have:

In W^n_1: $(probability\ of\ sum=i) = ps(i, n-1) \times (1 - p(t_n))$ (1)

Let us now consider W^n_2. The worlds involved in W^n_2 are constructed by adding t_n to each world of DB^{n-1}. Thus, for each sum value equal to i in DB^{n-1} we have a sum value equal to $(i + val(t_n))$ in W^n_2, where $val(t_n)$ is the aggr value of t_n. The probability of $sum = i + val(t_n)$ in W^n_2 is equal to the probability of $sum=i$ in DB^{n-1} multiplied by the membership probability of t_n. In other words, we have:

In W^n_2: $(probability\ of\ sum=i) = ps(i - val(t_n), n-1) \times p(t_n)$ (2)

Let $ps(i, n)$ be the probability of $sum=i$ in DB^n. This probability is equal to the probability of $sum=i$ in W^n_1 plus the probability of $sum=i$ in W^n_2. Thus, by using the Equations 1 and 2, and using the base of the recursion, we obtain the following recursive definition for the probability of $sum=i$ in DB^n, i.e. $ps(i, n)$:

$$ps(i,n) = \begin{cases} ps(i,n-1) \times (1 - p(t_n)) + ps(i - val(t_n), n-1) \times p(t_n) & if\ n > 1 \\ 1 - p(t_1) & if\ n = 1\ and\ i = 0\ and\ val(t_1) \neq 0 \\ p(t_1) & if\ n = 1\ and\ i = val(t_1)\ and\ val(t_1) \neq 0 \\ 1 & if\ n = 1\ and\ i = val(t_1) = 0 \\ 0 & otherwise \end{cases} \quad (4)$$

Based on the above recursive definition, we developed efficient algorithms for processing SUM and COUNT queries (see [2]).

3. Prototype

ProbDB is built on top of a classical Database Management System (DBMS). It adds probabilistic capabilities to the DBMS that are transparent to the user. Instead of directly modifying the DBMS and adding "native" primitives to it, we have chosen to implement ProbDB on top of the DBMS, and thus to be able to change the underlying DBMS with a slight programming effort. In its current version, the prototype is built atop PosgreSQL, but could easily be adapted to work on a MySQL database for instance.

When the user sends a query to ProbDB, the query is analyzed and probabilistic keywords are extracted. Then classical (non probabilistic) sub-queries are sent to the DBMS that process them and returns intermediate results. Then, probabilistic functions are applied to the intermediate results, and the final results are returned to the user.

ProbDB is composed of the following components (see the architecture in Figure 1):

Query parser: it is responsible for separating the probabilistic parts of the query from the ordinary ones. Let Q be the query given by the user. The parser divides Q to two parts: 1) Q': the subquery that can be evaluated by a deterministic DBMS ; 2) Q'': the parts of the query that need special probabilistic algorithms for being evaluated.

Query reformulator: this component reformulates the subquery Q' to a query that can be executed by the underlying deterministic DBMS over the data stored in the database. It needs the metadata of the probabilistic tables in order to translate each relation of Q' into one or more probabilistic relations in the database.

Deterministic DBMS: this is an ordinary (deterministic) relational database management system that given the reformulated Q', executes it over the probabilistic tables, and returns the results to the component that evaluates the probabilistic parts of the query.

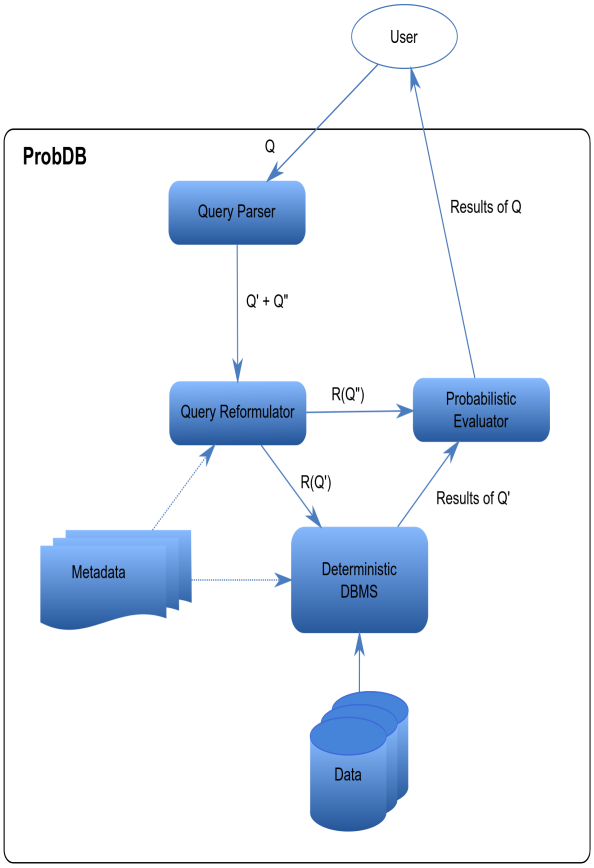

Figure 1: Architecture of ProbDB

Probabilistic evaluator. The inputs of this component are the intermediate data generated by the DBMS and the query Q''. According to the probabilistic expressions in Q'', the component chooses the appropriate algorithms and runs them over the intermediate results, and returns the final results to the user.

In ProbDB, we designed and implemented a user friendly graphical interface that allows the user to easily manipulate the data that are in attribute-level probabilistic model, query them and see the results of the queries visually (see Figure 2).

4. Application Scenarios

In our demonstration, we consider the data generated by the medical application described in Example 1. We use the relations designed for this application, and show the data manipulation and query processing facilities provided by ProbDB for the application. Particularly, we focus on the followings:

Metadata management. We show how ProbDB converts the data presented in a probabilistic attribute-level model to tables in a relational model. We show what happens in background when probabilistic data are inserted to the database. Figure 2 shows the interface for editing the probabilistic data in the database.

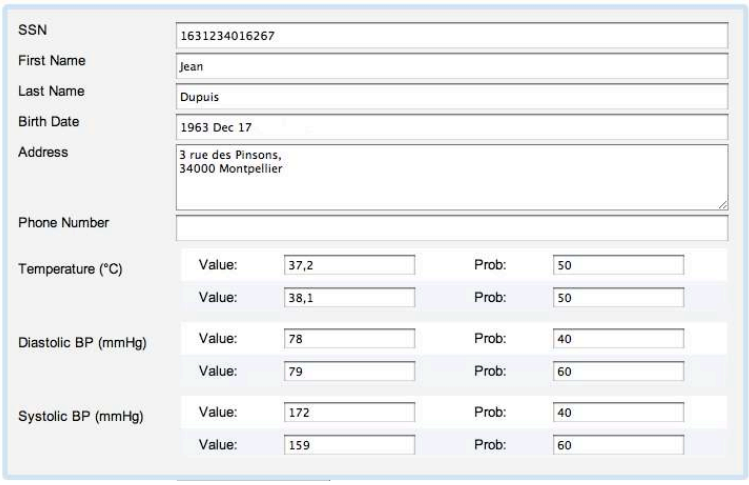

Figure 2: Editing patient's details

Fast query processing. In this scenario, we demonstrate how fast the probabilistic aggregate queries are processed in ProbDB. The user issues aggregate queries over the probabilistic health database, ProbDB returns quickly the aggregate results and their probabilities, and show them in a graphical user interface. In addition, the user can see in a diagram the cumulative distribution function (CDF) of the aggregate results returned by the system. This output is shown in Figure 3. The user can change the parameters of his/her query to see the impact on CDF. Using the CDF, we can answer queries such as how many nurses are needed to cover all emergency patients with a probability higher than 99%.

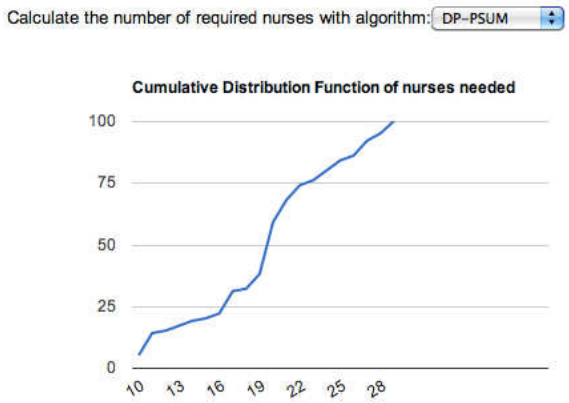

Figure 3: Results of the query

References

[1] A. Gal, M.V. Martinez, G.I. Simari and V. Subrahmanian. Aggregate Query Answering under Uncertain Schema Mappings. *In ICDE Conf.*, 2009.

[2] R. Akbarinia, P. Valduriez and G. Verger. SUM Query Processing over Probabilistic Data. *Research Report N°7629, INRIA*, France, 2011.