Towards Support for Uncertainty: MauveDB

Jun Yang
CPS 296.1, Spring 2007
Sensor Data Processing
With contents from A. Deshpande

Announcements (Mar. 6)

- This Thursday: project proposal talk
  - 15 minutes per group; 20% of total grade
  - What is it? Why do we care? Hasn't it been done before?
    Plans, thoughts, and preliminary results?
  - Submit your slides after class

Towards support for uncertainty

Motivation

- Unprecedented, and rapidly increasing, instrumentation of our every-day world
- Overwhelmingly large raw data volumes generated continuously
- Data must be processed in real-time
- Typically imprecise, unreliable and incomplete data

Traditional databases are ill-equipped to handle these challenges

Example: wireless sensor networks

A wireless sensor network deployed to monitor temperature

<table>
<thead>
<tr>
<th>Time</th>
<th>Sensor 1</th>
<th>Sensor 2</th>
<th>Sensor 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>10am</td>
<td>20</td>
<td>25</td>
<td>30</td>
</tr>
<tr>
<td>11am</td>
<td>22</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>12pm</td>
<td>20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SELECT time, AVG(temp) FROM Sensors EPOCH 1 HOUR

1. Spatially biased deployment ➔ These are not true averages!
2. High data loss rates ➔ averages of different sets of sensors
3. Measurement errors propagated to the user

Impedance mismatch

User wants to query the “underlying environment,” and NOT the sensor readings at selected locations

A wireless sensor network deployed to monitor temperature
Typical solution

- Process data using a statistical/probabilistic model
  - E.g., regression and interpolation
    - Eliminate spatial/temporal biases, handle missing data, predict
  - E.g., Kalman Filters, Bayesian Networks
    - To eliminate measurement noise, to infer hidden variables, etc.

Issues

- Databases typically only used as a backing store
- All data processing done outside!
- Processing is non-trivial
  - Expert knowledge & MATLAB familiarity may be required!
  - Lack of support for querying the processed data
  - Cannot exploit commonalities, reuse code, or share computation
    - Large amount of duplication of effort
  - No easy way to keep the model outputs up to date
    - Prevents real-time data analysis in most cases

Solution: model-based user views

- Abstraction analogous to traditional database views
- Provides independence from the messy measurement and modeling details

A traditional database view (defined using an SQL query) vs. A model-based database view (defined using a statistical model)
MauveDB system

- Supports the abstraction of Model-based User Views
- Provides declarative language constructs for creating such views
- Supports SQL queries over model-based views
- Keeps the models up-to-date as new data is inserted into the database

Example: linear regression

- Models a dependent variable as a function of a set of independent variables

Model temperature as a function of $x, y$, e.g.:

$$\text{temp} = w_1 + w_2 \times x + w_3 \times x^2 + w_4 \times y + w_5 \times y^2$$

Weights

Grid abstraction

A regression-based view
Defining a regression-based view

CREATE VIEW RegView(time [0::1], x [0:100:10], y [0:100:10], temp) AS
FIT temp USING time, x, y
BASES 1, x, x*x, y, y*y
FOR EACH time T
TRAINING DATA
SELECT temp, time, x, y FROM raw_temp_data WHERE raw_temp_data.time = T

View creation syntax

- Somewhat model-specific, but many commonalities
- E.g., an interpolation-based view:
CREATE VIEW IntView(t [0::1], sensorid [::1], temp) AS
INTERPOLATE temp USING time, sensorid
FOR EACH sensorid M
TRAINING DATA
SELECT temp, time, sensorid FROM raw_temp_readings WHERE raw_temp_readings.sensorid = M

Query a model-based view

Analogous to traditional views, e.g.: SELECT * FROM RegView;
- Lists out temperatures at all grid points
SELECT * FROM RegView WHERE x = 15 AND y = 20;
- Lists temperature at (15, 20) at all times
SELECT temp FROM IntView WHERE sensorid = 7 AND t = 100;
- Find the temperature at node 7 at time 100
Query processing

- Two operators
  - ScanView: returns the contents of a view tuple by tuple
  - IndexView(cond): return only tuples matching cond
    - E.g., return temperature where \((x, y) = (10, 20)\)

```sql
SELECT * 
FROM Locations l, RegView r 
WHERE  (l.x, l.y) = (r.x, r.y) 
AND r.time = 10;
```

View maintenance strategies

- No materialization: compute view as needed from base data
  - E.g., for regression view, scan the tuples and compute the weights
- Keep the view materialized
  - Sometimes too large to be practical (e.g., a fine grid)
  - May need to be recomputed with every new tuple (e.g., a regression view that fits a single function to the entire data)
- Lazy materialization/caching
  - Materialize query results as computed
  - Maintain an efficient intermediate representation
    - Typically model-specific

Intermediate rep. for regression

- Regression-based view
  - Training data \(\{(x_i, y_i, temp_i)\}, i = 1, \ldots, m\)
  - Regression model: \(w_1 b_1(x, y) + \ldots + w_k b_k(x, y)\)
  - Optimal \(w_i\)'s (that minimize root-mean-squared error) can be found by solving for \(w^*\) below:

\[
H = \begin{pmatrix}
  b_1(x_1, y_1) & \cdots & b_k(x_1, y_1) \\
  \vdots & \ddots & \vdots \\
  b_1(x_m, y_m) & \cdots & b_k(x_m, y_m)
\end{pmatrix} \\
\begin{pmatrix}
  temp_1 \\
  \vdots \\
  temp_m
\end{pmatrix} \\
H^T H w^* = H^T f
\]

- Maintain \(H^T H (k \times k)\) and \(H^T f (k \times 1)\)
  - Easy to update when new training data becomes available
Intermediate rep. for interpolation

- (Linear) interpolation-based view
  - Training data \( \{(t_i, v_i)\} \)
  - Given \( t \), find \( v \): search tree with \( t_i \) as key
  - Given \( v \), find \( t \): interval tree on \( \{[v_i-1, v_i)\} \)

Experiment data

- Intel Lab dataset
  - 54-node network
  - Attributes used: \( t, sensorid, x, y, \) temperature

Spatial regression
Interpolation

→ Over raw sensor readings

→ Over interpolated data

View maintenance options

- 50000 tuples initially
- Mixed workload
  - Insert 1000 records
  - Issue 50 point queries
  - Issue 10 average queries
- Intermediate representation typically the best
- Among others, dependent on the view properties and query workload

Discussion