Privatheit durch Anonymisierung von Anfragen an Datenbanken

NEidI/KSWS - WS 2015/2016

Gunnar Raßmann, Christian Langmacher, Irene Martin Rodriguez, Johannes Goltz

Agenda

1. Problemstellung
2. Aktueller Stand
3. Konzeption des JDBC-Treibers
4. Umsetzung des JDBC-Treibers
5. Zusammenfassung und Ausblick

Problemstellung

Einführung

- Big Data in Assistenzsystemen $\rightarrow$ Privacy in PArADISE umgesetzt
- Umsetzung für Einhaltung der Privatheit bei Sensordaten
- vertikale Architektur (spätere Umsetzung geplant)
- Sicherstellung gewisser Kenngrößen in Bezug auf Privatheit
- ohne Anonymisierung einfach Rückschlüsse aus Anfrageergebnissen auf einzelne Personen möglich
- keine Weitergabe von Daten aus eigenem Selbstbestimmungsraum, die Rückschluss auf das Individuum zulassen

Problemstellung

Umsetzung mit Hilfe von Datenbanken

- Auswertung von Anfragen über Datenbanken besonders effizient
- Anonymisierung der zurückgegebenen Ergebnisse wichtig
- Eingriff bei Anfragen im Treiber $\rightarrow$ Umsetzung der Anonymisierung
- Nutzung eines JDBC-Treibers und Neuschreiben der Funktionalitäten
- Möglichkeiten der Umsetzung von K-Anonymität, L-Diversität, T-Closeness und der Erkennung/Beseitigung von Quasi-Identifikatoren durch Generalisierung

Aktueller Stand

Probleme bei Nutzung des Standard-Treibers

- keine Anonymisierung vorgesehen
- durch bestimmte Attributkombinationen kann Personenbezug hergestellt werden
- Privatheit wird verletzt
- keine vertikale Architektur vorgesehen $\rightarrow$ alle Anfragen laufen über höchste Server-Instanz
- Ansatz: Schreiben eines eigenen JDBC-Treibers, um Anonymisierungstechniken umzusetzen

Aktueller Stand

Quasi-Identifikatoren und Generalisierung

- Quasi-Identifikator: Teilmenge von Attributen, die den Großteil der Tupel eindeutig identifizieren kann (schlüsselähnlich)
- Generalisierung: Anonymisierungstechnologie (Abbildung der Werte einer Spalte auf einen allgemeineren Wertebereich)

Aktueller Stand

K-Anonymität, L-Diversität, T-Closeness

- K-Anonymität: 1 Tupel von k-1 anderen nicht unterscheidbar
- L-Diversität: zusätzlich sensitive Werte in k-Gruppe (l unterschiedliche müssen auftauchen)
- T-Closeness: zusätzlich darf die Verteilung in der k-Gruppe hier höchstens um t von der Gesamtverteilung abweichen

Konzeption des JDBC-Treibers

Grobkonzept

- JDBC-Typ-3-Treiber komplett neu schreiben (Übertragung der JDBC-Aufrufe an Middleware in DBMS neutralen Format)
- größtmögliche Vielfalt bezüglich eingesetztem DBMS
- nativer Treiber für DBMS nur auf Serverseite nötig
- Nutzung einer Client-Server-Architektur (in JDBC-Treiber)
- Anonymisierung wird auf Client- und Serverseite umgesetzt
- Client erhält nur anonymisiertes ResultSet

Konzeption des JDBC-Treibers

Client

- fungiert als JDBC-Schnittstelle
- nimmt SQL-Query entgegen und passt es gegebenenfalls an
  - Transformation
  - Aufsplitten in mehrere Queries
- leitet Anfrage an den Server weiter
- nimmt (anonymisiertes) ResultSet entgegen
- bearbeitet erhaltenes ResultSet gegebenenfalls

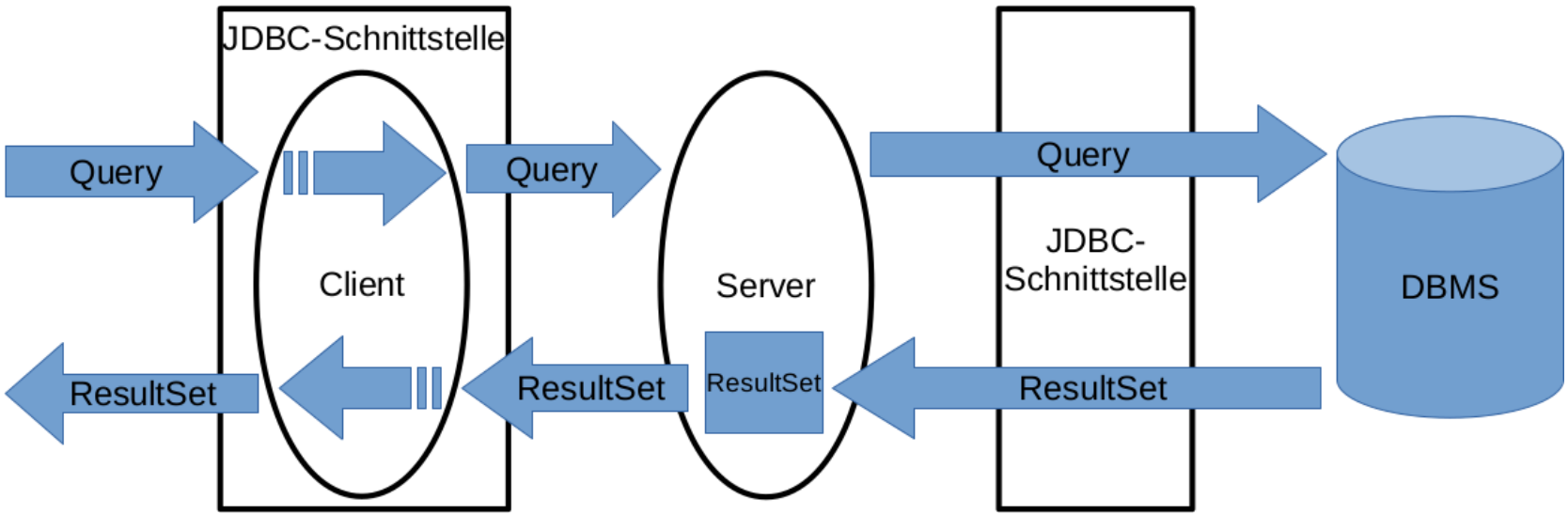
Konzeption des JDBC-Treibers

Server

- empfängt SQL-Query vom Client
- nutzt weitere JDBC-Schnittstelle um Query an DBMS zu senden
- erhaltenes ResultSet wird komplett zwischengespeichert
- ermittelt im ResultSet enthaltene Tabellen und Spalten
- anonymisiert das zwischengespeicherte ResultSet

Konzeption des JDBC-Treibers

Übersicht



Konzeption des JDBC-Treibers

Festlegung des Grades der Anonymisierung

- Speicherung der Werte in XML-Datei
  - K-Anonymity
  - L-Diversity
  - T-Closeness
  - Threshold-Werte für Quasi-Identifikatoren pro Tabelle
  - verwendetes DBMS und SQL-Version
- Nutzung von PP4SE (Privacy Policy for Smart Environments)

Umsetzung des JDBC-Treibers

Festlegung des Grades der Anonymisierung

- einlesen der XML-Datei
- XML auf Server als auch auf Client-Seite
  - Server besitzt gewisse Grundwerte
  - Verschärfung der Werte vom Server auf Client-Seite möglich (Abgleich erfolgt auf Server)
- eingelesene Daten werden in Java-Objekt geparst und zwischen Client und Server übertragen (mit entsprechenden Informationen für Anonymisierung)
- DBMS und SQL-Version wurden für spätere vertikale Architektur in das Objekt bereits mit aufgenommen

Umsetzung des JDBC-Treibers

PP4SE zur Festlegung der Anonymisierung

```
1   <modules>
2     <module module_ID="ThresholdQI" module_required="true" module_type="QI">
3       <module_description>All threshold QI—values for each table</module_description>
4       <attributeList>
5         <attribute name="adult">
6           <value>90</value>
7         </attribute>
8       </attributeList>
9     </module>
10    <module module_ID="Anonymity" module_required="true" module_type="Anonymity">
11      <module_description>Values for further anonymity of the data</module_description>
12      <kAnonymity>5</kAnonymity>
13      <lDiversity>2</lDiversity>
14      <tCloseness>45</tCloseness>
15    </module>
16    <module module_ID="Ability" module_required="true" module_type="Ability">
17      <module_description>Describes which DBMS is used and which Standard of SQL is implemented</module_description>
18      <dbms>MySQL</dbms>
19      <sql>SQL92</sql>
20    </module>
21  </modules>
```

Umsetzung des JDBC-Treibers

Query-Transformation

- Transformation der Query, sodass im ResultSet alle Informationen enthalten sind
- alle Spalten für Where-Klausel müssen im Select enthalten sein
- zu restriktive Where-Klausel auf QIs darf Query nicht ausgeführt werden (Dummy-ResultSet als Rückgabe)

Umsetzung des JDBC-Treibers

Query-Aufspaltung

- konzeptuelle Umsetzung
- Aufspaltung des Queries entsprechend der Fähigkeiten des Servers
- Nacheinanderausführen der Queries und Sammlung in ResultSet-Liste
- Kombination der ResultSets
- Rückgabe des konstruierten ResultSets

Umsetzung des JDBC-Treibers

Anonymisierung des ResultSets

- auch durch transformierte Queries produzierte ResultSets sind nicht vollständig unbedenklich
- Tupel könnten Personen (fast) eindeutig identifizieren
- auch aus nicht identifizierenden Attributkombinationen können Rückschlüsse gezogen werden
- $\rightarrow$ weitere Anonymisierung des ResultSets nach Berechnung

Quasi-Identifikatoren I

- QIs werden nach Tabellen aufgeteilt berechnet
- für jede Attributkombination muss lediglich der Quotient (SELECT COUNT DISTINCT (Attr))/(SELECT COUNT *) berechnet werden
- ist dieser Quotient höher als der Threshold der jeweiligen Tabelle, ist Attr ein QI

Umsetzung des JDBC-Treibers

Quasi-Identifikatoren II

- Berechnung mithilfe der performanten TopDownBottomUp-Methode
- zur Performance-Steigerung werden berechnete QIs in der Datenbank gespeichert und bleiben 12 Stunden aktiv
- interne Repräsentation: Liste von Objekten, die aus jeweils dem Tabellennamen und einer Liste von Attributmengen bestehen

Umsetzung des JDBC-Treibers

K-Anonymität

- Wert für k ist im Objekt bekannt
- verwenden Konkatenation von identifizierenden Spalten als Schlüssel
- zählen die Rows, die diesen Schlüssel enthalten
- wenn ein Schlüssel in weniger als k Zeilen enthalten ist, ist K-Anonymität verletzt

Umsetzung des JDBC-Treibers

Beispiel

age	**occupation**	**fnlwgt**
39	occ1-3	0-200000
50	occ1-3	0-200000
38	occ1-3	200001-400000
53	occ1-3	200001-400000
28	occ1-3	200001-400000
37	occ1-3	200001-400000
49	occ1-3	0-200000
52	occ1-3	200001-400000
31	occ1-3	0-200000
42	occ1-3	0-200000

Umsetzung des JDBC-Treibers

L-Diversität

- Wert für l ist im Objekt bekannt
- verwenden Konkatenation von Attributwerten der QIs als Schlüssel
- zählen die unterschiedlichen Werte zu jedem Schlüssel
- wenn zu einem Schlüssel weniger als l Werte auftauchen, ist L-Diversität verletzt

Umsetzung des JDBC-Treibers

Beispiel

age	**occupation**	**fnlwgt**
39	occ1-3	0-200000
50	occ1-3	0-200000
38	occ1-3	200001-400000
53	occ1-3	200001-400000
28	occ1-3	200001-400000
37	occ1-3	200001-400000
49	occ1-3	0-200000
52	occ1-3	200001-400000
31	occ1-3	0-200000
42	occ1-3	0-200000

Umsetzung des JDBC-Treibers

T-Closeness

- wählen QIs als Schlüssel
- wählen den Vektor aller numerischen Nicht-QI-Spalten als Wert
- berechnen zu jedem Schlüssel den Mittelwert, sowie den Mittelwert über alle Tupel
- iteratives Verfahren, darum nur ein Durchlauf über alle Tupel
- berechnen die Distanz der Mittelwerte zum Gesamtmittelwert (derzeit Manhattan)
- ist eine Distanz größer als t, ist T-Closeness verletzt

Umsetzung des JDBC-Treibers

Beispiel

age	**occupation**	**fnlwgt**
39	occ1-3	0-200000
50	occ1-3	0-200000
38	occ1-3	200001-400000
53	occ1-3	200001-400000
28	occ1-3	200001-400000
37	occ1-3	200001-400000
49	occ1-3	0-200000
52	occ1-3	200001-400000
31	occ1-3	0-200000
42	occ1-3	0-200000

Umsetzung des JDBC-Treibers

Workflow

- beim Erstellen einer Connection: Berechnung/Laden der QI und Laden der DGHs aus Datenbank
- Berechnung mithilfe einer nicht anonymisierten Funktion, die auch für Lernalgorithmen verwendet werden kann
- Aufruf einer Anonymisierungsfunktion auf dem, durch das transformierte Query, erzeugten ResultSet

Umsetzung des JDBC-Treibers

Workflow: Anonymisierung I

```
1 HashMap<String,Set<String>> pColumns=GetQiInQuery(
      effectedColumns);
2
3 if(!pColumns.isEmpty()){
4         _encryptedColumns=AddToPrivateColumns(pColumns
              ,_encryptedColumns);
5         resultSet=ApplyPrivacyRules(resultSet,pColumns
              );
6 }
7 else return resultSet;
```

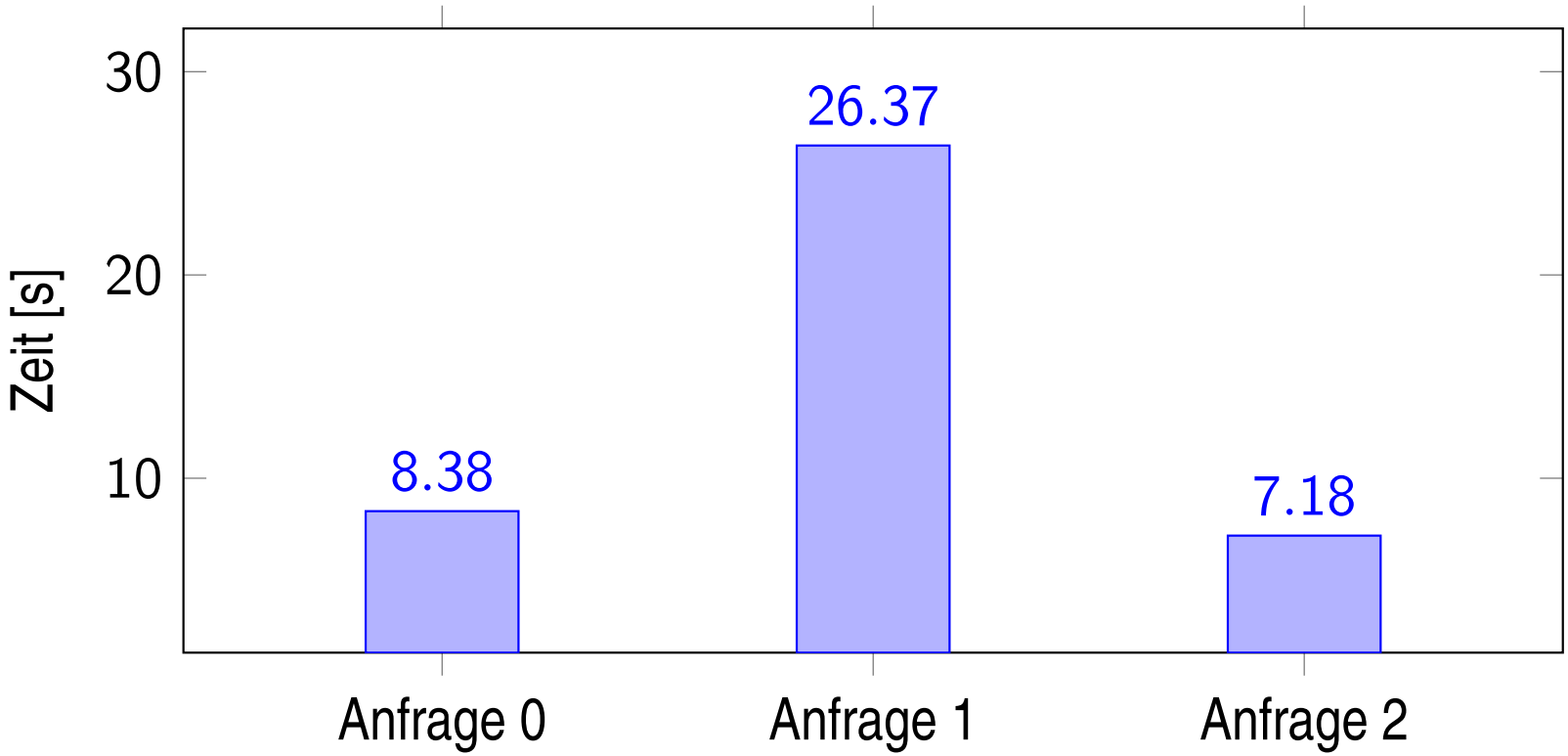
Umsetzung des JDBC-Treibers

Workflow: Anonymisierung II

```
1 while(breaks){
2         HashMap<String,Set<String>> thispColumn=
              GetNewEncryptedColumn(resultSet,pColumns);
3         resultSet=ApplyPrivacyRules(resultSet,
              thispColumn);
4         breaks = BreaksPrivacy(resultSet,
              effectedColumns);
5         _encryptedColumns=AddToPrivateColumns(
              _encryptedColumns,thispColumn);
6 }
```

Umsetzung des JDBC-Treibers

Vergleich Anonymisierungsdauer

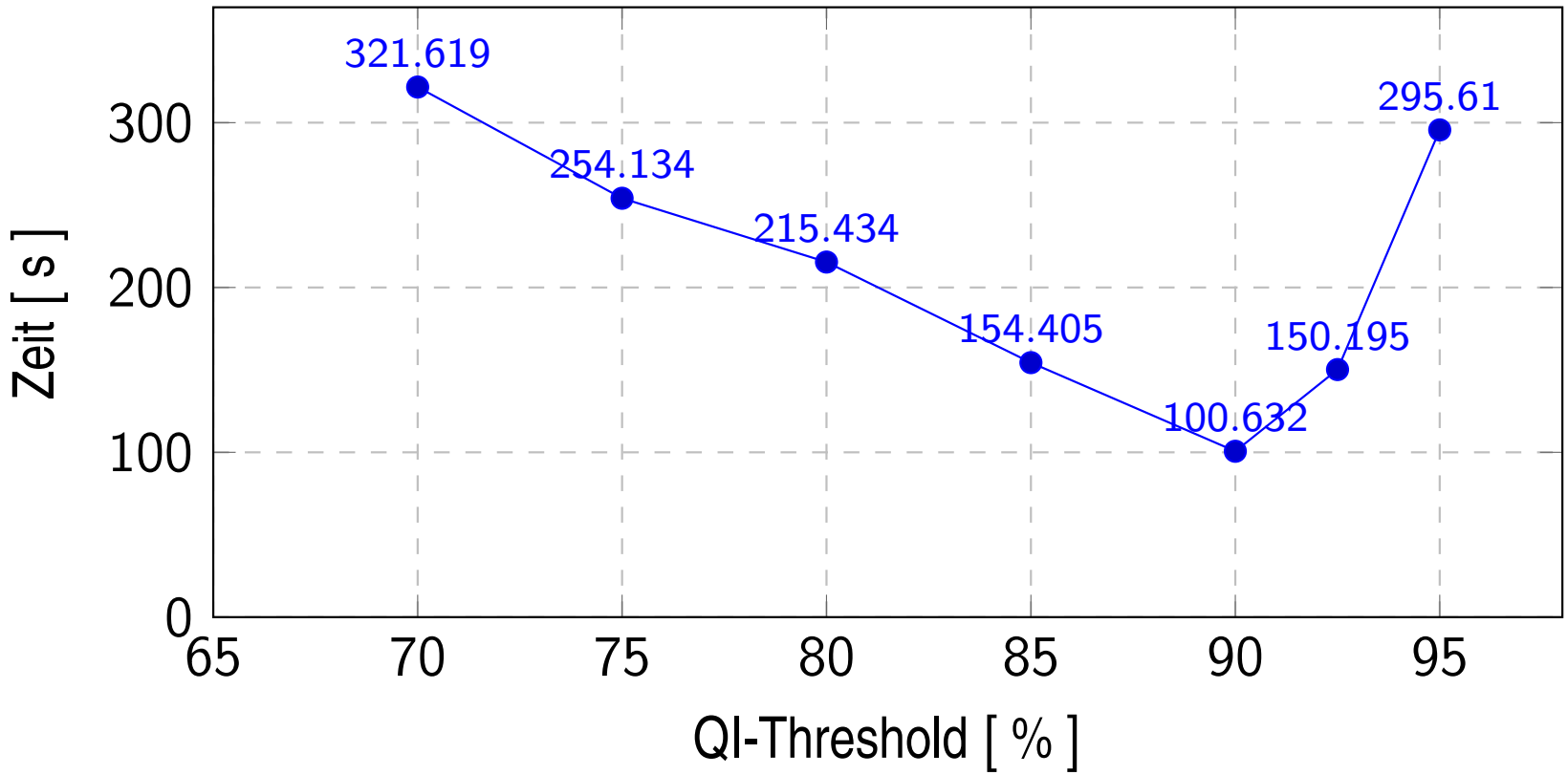


Anfrage 0: 3 Spalten (2 QIs), Anfrage 1: 10 Spalten (10 QIs), Anfrage 2: 3 Spalten (kein QI)

Jeweils 30'000 Zeilen und Messung per time (real-time)

Umsetzung des JDBC-Treibers

Vergleich QI-Berechnung



3 Spalten angefragt, kein QI, 1 Zeile; Messung per time (real-time); Anfragedauer etwa 2s

Zusammenfassung und Ausblick

Funktionen des Treibers

- lesende Anfragen an DBMS
- einfache Datentypen im ResultSet
- Anonymisierung des ResultSets
  - K-Anonymität
  - L-Diversität
  - T-Closeness (für numerische Werte)

Zusammenfassung und Ausblick

Weiterführende Arbeit

- Update-Operationen in der Datenbank
- komplexere Datentypen (z.B. Streams, BLOBs, ...)
- Transformation der SQL-Anfrage zurzeit nur konzeptionell
- Hintereinanderschachtelung unseres Treibers
  - benötigen Update-Operationen für QIs
  - DGHs und QIs dürfen nicht anonymisiert gelesen werden

Fragen

Vielen Dank für die Aufmerksamkeit!