Conflict Tolerant Queries in AURORA

Ling Ling Yan and M. Tamer Özsu
Laboratory for Database Systems Research
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2H1
{ling, ozsu}@cs.ualberta.ca

Abstract

Conflict tolerant queries are a new way of dealing with instance level conflicts in data integrated from multiple sources. In contrast to the traditional approach of resolving such conflicts during schema integration using aggregation functions, we establish a query model and processing techniques to tolerate these conflicts at query time to a degree specified by the users. Resolutions are only performed to produce conflict-free results. Currently we support 3 levels of conflict tolerance: HighConfidence, RandomEvidence, and PossibleAtAll and allow user-defined functions to be used for conflict resolution. The approach reduces the overhead of conflict detection and resolution and lends itself to new query optimization techniques. Fundamentally, our approach allows users to handle conflict at a coarse granularity to achieve better query performance when conflict resolution requirements are relaxed and when data contain occasional conflicts.

1. Introduction

Vast amount of digital information is stored in a variety of data sources. With the advent of the Internet, the way people use information is changing rapidly; integrated access to heterogeneous sources is required. When integrating data from such sources, two types of conflicts may arise: semantic conflicts, which happen when sources model the same application differently, and instance level conflicts, which happen when sources record inconsistent values on the same objects. In this paper, we propose a technique for querying data in the presence of instance level conflicts. Traditionally, these conflicts are resolved at schema integration time using aggregation functions [2]. For instance, one may specify that when multiple sources record different age values for a person, the “correct” age be computed as the average of these values. Queries are written as if data are conflict-free. Conceptually, instance level conflicts are resolved before queries are evaluated; users have no say over resolution policies at query time. We refer to this approach as the static resolution approach. These resolutions are realized during materialization or query processing. If integrated data are materialized, instance level conflicts are removed before any query is processed. If data are virtual, enough data must be retrieved for conflict detection and resolution at query time; this may incur significant performance penalty as illustrated below:

**Example 1.1** Assume that sources A and B provide data on Person and conflicts on Age are to be resolved by taking the average of all Age values. Consider query:

\[ Q_0 = \text{select ID, Name, Address from Person where Age} > 30 \]

It is not sufficient to retrieve only persons with \(\text{Age} > 30\); we must retrieve all Person data from both A and B, compute all Age values, and evaluate the query. This cost stays the same even when no Age conflict actually occurs. Optimization strategies have been proposed but cases such as \(Q_0\) are fundamentally difficult to optimize. This drawback becomes significant when more sources contribute large volumes of Person data.

In a dynamic data integration system where large numbers of data sources come and go, materialization may not be desirable. It may also be difficult to foresee when and where instance level conflicts are likely to happen; adding a new source may give rise to new conflicts. Specifying a resolution for conflicts that do not really happen incurs unnecessary performance penalties if data are virtual. On the other hand, applications vary in requirements for conflict handling. For \(Q_0\) in Example 1.1, the exact age of a person does not matter so long as he/she is older than 30. When multiple sources offer different Age values of a person, one user may consider him to be older than 30 if some sources say so, while another may require that all sources
say so. Conflict resolutions on Name and Address can be performed only for persons who qualified as older than 30. Conflicts on Person.Age are not resolved, but rather tolerated by the system during query processing. We refer to this approach of instance level conflict handling conflict-tolerant (CT) querying.

<table>
<thead>
<tr>
<th>conflicts Evaluation</th>
<th>Statically Resolved</th>
<th>Tolerated</th>
</tr>
</thead>
<tbody>
<tr>
<td>On Materialized Data</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>On Virtual Data</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

**Table 1. Querying Integrated Data**

Depending on whether integrated data are materialized and how instance level conflicts are handled, we distinguish among 4 cases of querying integrated data, as shown in Table 1. Cases 1 and 2 raise no new issues in query semantics; these are well-studied domains. Case 1 requires maintenance of materialized data. Query optimization issues in case 2 has been studied. The CT query model applies to cases 3 and 4. Optimizing queries on materialized data leverages existing techniques and is not discussed.

The CT query model enables users to resolve instance level conflicts to a desired degree and let the system "tolerate" the rest; it allows flexible conflict handling and better query performance for users who do not require static resolutions. Consider the following CT query:

\[
Q_0 = \text{select PIN, Name[ANY], Age[ANY], Address[DISCARD]} \\
\text{from Person} \\
\text{where Age > 30 with HighConfidence}
\]

*HighConfidence* in the “with” clause specifies that if inconsistent age values exist, a person qualifies as *Age* > 30 only if *all* sources say so. After a person qualifies, if there is conflict on Name, Age, or Address, the functions ANY, ANY, and DISCARD, respectively, are used to remove these conflicts to produce a conflict-free query answer. Given a set of values *S*, function ANY returns a random value from *S*, function DISCARD returns a null value if *S* contains more than one distinct value, otherwise it returns the only value in *S*. These resolutions do not affect predicate evaluation; they are only used to construct conflict-free query results. If all sources record that Fred is younger than 30, then he does not have to be retrieved even if there is conflict on his Age. The framework described in Section 6 enables such optimized processing.

The CT query model and its optimization framework are developed as part of AURORA system [8], which provides a mediation framework for scalable integration of large number of sources. Section 2 describes the relevant aspects of AURORA. Section 3 defines conflicts and resolutions in AURORA terms. Section 4 defines the CT query model. Section 5 describes primitive CT query evaluation. Section 6 describes a query optimization framework. Section 7 reviews related work. Section 8 contains conclusions and future work.

![Figure 1. The AURORA Mediation Framework](image)

**Notation.** \( t.A \) denotes the value of attribute *A* in tuple *t*, and \( R[A] \) denotes all values of attribute *A* in relation *R*. Given a collection of relations, \( Y = \{F_1, ..., F_m\} \), and an attribute \( B, Y[B] = F_1[B] \cup ... \cup F_m[B] \). \( ATT(R) \) is the set of attributes of relation *R*. \( ATT(R)(p) \) is the set of attributes referenced by predicate *p*.

2. Data Integration in AURORA

AURORA employs a two-tiered, plug-and-play mediation model depicted in Figure 1. This model is designed to facilitate dynamic and scalable data integration [8].
Sources are first homogenized and then integrated. Homogenization removes idiosyncrasies of individual sources, which is done independently and, possibly, in parallel. A homogenized source describes its content to the integration mediator to which it contributes data. The integration mediator deals with a large number of homogenized sources and is fully automatic. All mediators are data model specific. Currently the system has relational and object-oriented mediators. This paper considers only the relational ones. Figure 2 shows the current architecture of AURORA. AURORA-RH [10] is the Relational Homogenization mediator and AURORA-RI is the Relational Integration mediator. Wrappers support relational interface to sources. In the current implementation, we use OLE-DB providers supporting SQL as wrappers, such wrappers are readily available for a large variety of data sources. Mediators and wrappers cooperate via COM/DCOM.

The integration mediator, AURORA-RI, maintains a predefined service view, a usual relational schema that can be queried by applications. Sources wishing to participate in the service view $S$ maintained by an AURORA-RI mediator $M$ must be homogenized against $S$ using an AURORA-RH mediator, which also communicates with $M$ to describe the content of the source in the context of the service view. At query time, AURORA-RI merges data from relevant sources and deals with instance level conflicts using the C/T query model.

### 2.1. Service View

For applications, the service view is a relational schema that can be queried. For sources that provide data through this view, it is a pre-defined relational schema where each relation, called a global relation, specifies a group of attributes as its plug-in identifier (PID). The PID is used by AURORA-RI for object matching, to identify tuples from different sources that describe the same object so that they can be combined to form tuples in the global relation; a source tuple must carry relevant PID in order to “fit” into the service view. Intuitively, the PID is a “ticket” to the service view. We use $PID(R)$ to denote the PID of relation $R$. To simplify presentation, we also assume that $PID(R)$ is a single attribute. For $t \in R$, its PID value is denoted as $t.PID$. For example, a service view may contain relation Person described below with $PID(Person) = “ID”: Person(ID, Name, Age, HomeNo, WorkNo, Employer, NoWorkYear, NoSchoolYear)

### 2.2. Registrations, Fragments, and Match Join

A data source must register the data it provides to a target AURORA-RI mediator. A registration is a 3-tuple:

$$REG = \langle DSN, SR, GRN >$$

where $DSN$ is the data source name, $SR$ is a source relation schema, and $GRN$ is a global relation name. Once this registration goes through, $SR$ becomes a registered fragment of $GRN$. The attribute set of $SR$ must include the PID of $GRN$, that is, $PID(GRN) \subseteq ATTR(SR)$. For any attribute $B \in ATTR(GRN)$, if $B \in ATTR(SR)$, we say that source relation $SR$ supports $B$. A registered fragment of a global relation often supports some, but not all, of its attributes.

AURORA-RI uses the Match Join (MJ) operator to “manufacture” tuples in global relations using registered fragments based on PID values. Consider two registered fragments, $F1(P, A, B, C)$ and $F2(P, B, C)$, of global relation $R(P, A, B, C)$ with PID $P$. If $p, a, b, c \in F1$, $p, a, b, c \in F2$, then $p, a, b, c \in R$. If $p, a, b, c \in F2$ and $b \neq b'$, then both $p, a, c, b' \in F1$ and $p, a, b', c > a, b' > p, a, b, c > p$. MJ can be expressed using outer-joins.

**Definition 2.1** Let $Y = \{F_1, ..., F_M\}$ be a set of fragments with a common PID $P$. Let $A_i$ be a non-PID attribute. The value set of $A_i$ given $Y$ is defined as:

$$VALset(A_i|Y) = \bigcup_{j=1}^{M_i} \pi_{P.A_i}(F_{ij})$$

where $F_{ij}$’s (1 $\leq j \leq M_i$) are all the fragments in $Y$ supporting $A_i$. □

$VALset(A_i|Y)$ is a binary relation $(P, A_i)$ containing all the $A_i$-values from the fragments in $Y$ and the related PID value. These binary relations are then outer-joined to derive a global relation.

**Definition 2.2** Let $Y = \{F_1, ..., F_M\}$ be a set of fragments with a common PID $P$. Let $S = \{P, A_1, ..., A_g\}$ be a set of attributes, $\forall 1 \leq i \leq g, A_i \neq P$. The Match Join (MJ) of relations in $Y$ based on $P$ in regard to $S$ is defined as:

$$MJ(P, S, Y) = VALset(A_1|Y)\pi_PVALset(A_2|Y)\pi_P...\pi_PVALset(A_g|Y)$$

where $\pi_P$ denotes outer-equi-join on $P$. □

Let $R$ be a global relation and let $Y_R$ be the set of all fragments registered with $R$. $Y_R = \{F_1, ..., F_M\}$. Then relation $R$ is derived as:

$$R = MJ(PID(R), ATTR(R), Y_R).$$

Global relations thus computed may contain null values. For any tuple $t$ and predicate $p$, we assume that $p(t)$ is true if and only if $t$ contains no null values on all attributes referenced by $p$ and $p$ holds on $t$.

**Example 2.1** Assume that at one mediator we have defined a global relation Person as:

Person(ID, Name, Age, HomeNo, WorkNo, Employer, NoWorkYear, NoSchoolYear)

with PID “ID”. Also assume that Person has four registered fragments, as shown in Figure 3. AURORA-RI will derive Person as shown in Figure 4. The column $tid$ is not part of the result but is used later to refer to tuples. □
3. Instance Level Conflicts and Resolutions

In Figure 3, Fragment 1 records that Fred is 32 years old while Fragment 4 indicates that Fred's age is 34. This conflict is reflected in Figure 4 as a violation of key constraint, since there are more than one tuple with ID 003; these tuples form the **alternative tuple set** for 003, denoted as $ATset(Person, 003)$. An alternative tuple set containing more than one tuple with ID 003 indicates an instance level conflict.

Formally, for global relation $R$ and a PID value $k$. The alternative tuple set of $R$ at $k$, $ATset(R, k)$, is defined as:

$$ATset(R, k) = \{t \mid t \in R, t.PID = k\}$$

For example, we have the following in Figure 4:

- $ATset(Person, 001) = \{t_1\}$
- $ATset(Person, 002) = \{t_2, t_3, t_4, t_5\}$
- $ATset(Person, 004) = \{t_{14}, t_{15}\}$

If $|ATset(R, k)| \geq 2$, we say there is a **conflict** in $R$ at $k$. Relations that may contain conflicts are called conflict-accommodating relations, or CA-relations.

Global relations derived using MJ are CA-relations; this is because we make no effort to remove any conflicts during this derivation. Conflicts are caused by inconsistencies among registered fragments and demonstrate themselves as $ATsets$ with cardinalities larger than 1. $ATset$ describes conflicts at tuple level. These conflicts are caused by one or more conflicts at attribute level, For global relation $R$, non-PID attribute $A$ and PID value $k$, we say there is an **attribute level conflict** on $R.A$ at $k$ if $|ATset(\pi_{PID}, A(R)), k)| \geq 2$.

Even with the conflict tolerant query model, conflict resolution must still be performed, although delayed and relaxed in a controlled way. AURORA provides two operators, Resolve Tuple level Conflict (RTC) and Resolve Attribute level Conflict (RAC), for conflict resolution at tuple and attribute levels, respectively. These operators are used for defining the CT query model later. Both operators take a resolution as a parameter. As defined below, a resolution is a function with an appropriate signature; it can be system-defined or provided by users.

**DEFINITION 3.1** Given a global relation $R$ and its attribute $A$, an attribute conflict resolution on $R.A$ is a function $\text{f:} setof(T) \rightarrow T$, where $T$ is the type of $R.A$. A tuple conflict resolution on $R$ is a function $g$ such that, given a set of tuples $S = \{t_1, ..., t_n\} \subseteq R.t.PID = k$ for $1 \leq i \leq n$, $g(S) = t$ where $\text{ATTR}(t) = \text{ATTR}(R), t = \text{mul}$ or $t.PID = k$. □

AURORA provides common resolution functions such as SUM, AVG, MAX, MIN, ANY, DISCARD, but also allows user-defined functions. If we resolve conflicts on all attributes, we effectively have defined a tuple conflict resolution. This relationship between the two types of resolutions is captured by the concept of **equivalent tuple conflict resolution** (ETCR) given below. This concept allows us to build the CT query model only based on tuple-level conflict resolutions, although the users can still specify attribute level conflict resolutions if they wish. Traditionally, attribute level conflicts are the only type of conflicts discussed [2]; users may be more comfortable with them.

**DEFINITION 3.2** Let $R$ be a global relation and $X = \{A_1, ..., A_n\}$ be all the non-PID attributes of $R$ over which there may be conflicts. Let $f_1, ..., f_n$ be attribute conflict resolutions on $A_1, ..., A_n$, respectively. Let $S$ be a set of tuples of $R$ that have the same PID value. A tuple conflict resolution of $R, g$, is the **Equivalent Tuple Conflict Resolution (ETCR)** of $f_1, ..., f_n$, denoted as $g = ETCR(f_1, ..., f_n)$, if for any set of $R$-tuples with a common PID value, $S$, $g(S) = t$ where $t$ satisfies the following:

<table>
<thead>
<tr>
<th>Fragment 1</th>
<th>Fragment 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>001</td>
<td>Peter</td>
</tr>
<tr>
<td>002</td>
<td>Mary</td>
</tr>
<tr>
<td>003</td>
<td>Fred</td>
</tr>
<tr>
<td>004</td>
<td>James</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fragment 3</th>
<th>Fragment 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>WorkNo</td>
</tr>
<tr>
<td>002</td>
<td>2000</td>
</tr>
<tr>
<td>003</td>
<td>3003</td>
</tr>
<tr>
<td>004</td>
<td>4000</td>
</tr>
<tr>
<td>005</td>
<td>5005</td>
</tr>
<tr>
<td>006</td>
<td>6000</td>
</tr>
<tr>
<td>007</td>
<td>Julie</td>
</tr>
<tr>
<td>008</td>
<td>Alan</td>
</tr>
</tbody>
</table>

**Figure 3. Registered Fragments for Global Relation Person**
1. \( \forall i, 1 \leq i \leq n, t.A_i = f_i(S_i) \text{ where } S_i = \{ v \mid \exists r \in S_i, r.A_i = v \} \); and

2. \( \forall B \in \text{ATTR}(R) - X, t.B = r_1.B \), where \( r_1 \in S \).

\[ \square \]

**Definition 3.3** Let \( R \) be a CA-relation and \( f_1, ..., f_n \) be conflict resolutions on non-PID attributes \( A_1, ..., A_n \). Operator Resolve Attribute Conflict, \( RAC \), is defined as

\[
RAC(R, A_1; f_1, ..., A_n; f_n) = \{ t' \mid \exists k, t \in R \ni t.PID = k, t'.PID = k, \\
\forall i, 1 \leq i \leq n, t'.A_i = f_i(S(R, A_i, k)), \forall B \in \text{ATTR}(R) - \{ A_1, ..., A_n \}, t'.B = t.B \}
\]

where \( S(R, A, k) = \{ a \mid < k, a \in \pi_{\text{PID}}(R) \} \). \( \square \)

**Definition 3.4** Let \( R \) be a CA-relation and \( F \) be a tuple conflict resolution of \( R \). Operator Resolve Tuple Conflict, \( RTC \), is defined as

\[
RTC(R, F) = \{ t \mid \exists k, t = F(\text{ATTR}(R, k)) \}
\]

\[ \square \]

Intuitively, \( RAC \) removes conflicts on attributes \( A_1, ..., A_n \) of \( R \) using functions \( f_1, ..., f_n \). \( RTC \) removes tuple level conflicts using function \( F \). These operators are illustrated in Figures 5 and 6. Given \( C \), a set of conflict resolution functions for all the non-PID attributes of \( R \) over which there may exist conflicts, we have \( RTC(R, ETCSR(C)) = RAC(R, C) \).

### 4. Conflict Tolerant Query Model

We define the semantics of single relation CT queries. A CT query over global relations \( R_1, ..., R_n \) is semantically equivalent to a single relation CT query over \( R_Q = R_1 \times ... \times R_n \). The PID of \( R_Q \) includes PIDs of all involved relations.

Single relation CT queries are in the following form:

\[
Q_{CT} = \text{SELECT } L \text{ FROM } R \text{ WHERE } p \text{ WITH } c_1
\]

where \( L \) is in one of the following forms:

1. \( L = E_1, ..., E_m \text{ where } E_i = R_i.B_i (1 \leq i \leq m) \text{ if } B_i \neq PID(R) \text{ if } B_i \neq PID(R), \) \( d_i \) is an attribute conflict resolution for \( R_iB_i \).

2. \( [D]R_1B_1, ..., R_mB_m \text{ where } D \text{ is a tuple conflict resolution for } \pi_{\text{PID}}(R_1B_1, ..., R_mB_m) \).

\( c_1 \) is called the predicate evaluation parameter, or PE-parameter, \( c_1 \in \{ \text{HighConfidence, RandomEvidence, PossibleAtAll} \} \); \( d_i \)'s and \( D \) specify how conflicts are removed to produce a conflict-free query answer. \( Q_1 \) and \( Q_2 \) are example CT queries:

\[
Q_1 : \text{SELECT } \text{PIN, Name[ANY], Age[ANY], Address[DISCARD]} \text{ FROM Person} \text{ WHERE Age > 30 WITH HighConfidence}
\]

\[
Q_2 : \text{SELECT } \text{[ANY] PIN, Name, Age, Address FROM Person WHERE Age > 30 WITH RandomEvidence}
\]
Both queries retrieve PIN, Name, Age and Address of persons older than 30. When there is conflict on Age, \(Q_2\) selects persons for whom all Age values available are \(>30\), while \(Q_2\) randomly sample one Age value and if it is \(>30\), then the person is selected. After a person qualifies as \(>30\), there may still be conflicts on Name, Age or Address; these conflicts are resolved using the resolutions specified in the select clause. \(Q_1\) resolves conflicts on attribute level while \(Q_2\) does it on tuple level. We support a few default forms of \(L\). \(L = A_1, ..., A_n\), where \(A_3\) are attributes, is the same as \(L = \text{ANY}[A_1, ..., A_n]\). If at least one attribute resolution is specified in \(L\), the default resolution for all other non-PID attributes with no specified resolution is ANY. Fundamentally, no matter which form \(L\) takes, it specifies a tuple conflict resolution, \(DE(L)\), referred to as the data extraction parameter, the DE-parameter. If \(L\) is in form 2, \(DE(L) = D\). A form 1 select clause can be rewritten into form 2 with \(D = ETCR(d_1, ..., d_m)\). We only consider form 2 select clause in the rest of the presentation.

Semantics of \(Q_{CT}\) is defined in two steps. First, we define how to find all the PID values that identify objects in \(R\) that satisfy the query predicate \(p\); this set is called the contributing PID set \((CSET)\). Since \(R\) is a CA-relation, a given PID value, \(k\), may identify a set of tuples, namely \(ATset(R, k)\). Whether \(k\) identifies an object that satisfies \(p\) must be determined by properties of \(ATset(R, k)\) against \(p\) given PE-parameter \(c_1\). Second, we have to remove any conflicts on data related to PIDs in the CSET computed earlier according to the DE-parameter of \(Q_{CT}\). Semantics of \(Q_{CT}\) is formally defined below.

**Definition 4.1** Given a CA-relation \(R\), a predicate \(p\) and a PE-parameter \(c_1\), the contributing PID set \((CSET)\) of \(R\) in regard to \(p\) under \(c_1\), \(CSET(R, p, c_1)\), is defined as follows:

1. For any \(k \in R\{PID\}\) such that \(|ATset(R, k)| = 1\), \(k \in CSET(R, p, c_1)\) if and only if \(p(t) = true\), where \(t \in ATset(R, k)\).

2. For any \(k \in R\{PID\}\) such that \(|ATset(R, k)| \geq 2\):
   - If \(c_1 = \text{RandomEvidence}\), \(k \in CSET(R, p, c_1)\) if and only if \(p(t) = true\), where \(t \in ATset(R, k)\) is selected by a function at query evaluation time.
   - If \(c_1 = \text{PossibleAtAll}\), \(k \in CSET(R, p, c_1)\) if and only if \(\exists t \in ATset(R, k)\), \(p(t) = true\).
   - If \(c_1 = \text{HighConfidence}\), \(k \in CSET(R, p, c_1)\) if and only if \(\forall t \in ATset(R, k)\), \(p(t) = true\).

\(\Box\)

A \(CSET\) contains PIDs identifying tuples that satisfy a predicate under a given PE-parameter; these tuples will contribute to the query result. When the PE-parameter is \(\text{RandomEvidence}\), the value of \(CSET\) depends on the run-time function used to choose a tuple from an \(ATset\) based on which the query predicate is evaluated. Thus more than one \(CSET\) can be considered valid. Such variations are captured by the following definition.

**Definition 4.2** Let \(R\) be a CA-relation, \(p\) a predicate and \(c\) a PE-parameter. A set of PID values \(C\) is a valid \(CSET\) of \(R\) in regard to \(p\) under \(c_1\) if:
\* \* c \neq \textit{RandomEvidence} and \( C = \text{CSET}(R, p, c) \); or
\* \* c = \textit{RandomEvidence} and \( \forall k \in C \), such that \( k \notin \text{CSET}(R, p, \text{HighConfidence}) \), there exist tuples \( t_1, t_2 \in R \), such that \( t_1, \text{PID} = t_2, \text{PID} = k \), \( p(t_1) = \text{false}, p(t_2) = \text{true} \).
\* \* \* \*  

\begin{algorithm}
\caption{CT-QP-NoOpt} \label{alg:ct-qp-noopt}
\begin{algorithmic}
\\textbf{input:} \( R \): Global relation involved in the query.
\( Q \): \( Q = \text{select } I \text{ from } R \text{ where } p \text{ with } c_1 \).
\( F_i \): All the fragments registered with \( R, F_1, \ldots, F_n \).
\end{algorithmic}
\begin{algorithmic}[1]
\end{algorithmic}
\end{algorithm}

\[ A = \pi_{B_1, \ldots, B_m} [\text{RTC}(R \mid \text{PID} \in \text{CSET}(R, p, c_1), \text{DE}(L))] \]
\* \* \* \*  

Table 2 shows 12 CT queries and results. These queries vary in PE-parameter and DE-parameter. We use two DE-parameters: ANY and DISCARD but they can be any function defined by the system or user. We vary the select clause to demonstrate how CT query model tolerates conflicts. Results of queries involving \textit{RandomEvidence} or ANY may vary with the selection function used at run-time. By specifying these parameters, one accepts such variations.

\begin{example}
First examine \( Q_1, Q_2 \) shown in the left columns of Table 2. The most stringent control appears in \( Q_2 \). This query has one of the smallest results. We next observe that queries in the 3rd column often have larger results than those in the 1st column. For example, \( Q_4 \) and \( Q_5 \). This is because relation Person contains no conflicts over \textit{Name} but it contains conflicts over \textit{Age}. When a query retrieves only conflict-free attributes, conflicts on other attributes are often hidden from the users altogether; the system does not resolve conflicts on them either. □
\end{example}

5. Primitive CT Query Evaluation

Algorithm CT-QP-NoOpt is an unoptimized algorithm that directly implements the CT query semantics given earlier. Correctness of this algorithm is straightforward.

\begin{algorithm}
\caption{CT-QP-NoOpt} \label{alg:ct-qp-noopt}
\begin{algorithmic}
\\textbf{input:} \( R \): Global relation involved in the query.
\( Q \): \( Q = \text{select } I \text{ from } R \text{ where } p \text{ with } c_1 \).
\( F_i \): All the fragments registered with \( R, F_1, \ldots, F_n \).
\end{algorithmic}
\begin{algorithmic}[1]
\end{algorithmic}
\end{algorithm}

6. Optimizing CT Query Processing

For predicate \( p \) over a global relation \( R \) and a fragment of \( R, F \), if \( \text{ATTR}(p) \subseteq \text{ATTR}(F) \), we say \( p \) is applicable to \( F \). CT query optimization aims at using applicable predicates to reduce the volume of fragment data retrieved into the mediator while preserving query semantics.
Table 2. Example Queries and Answers

6.1. CT Query Optimization Examples

Let $p$ be a predicate over $R$ and let $p = p_1 \land \ldots \land p_m$ be its conjunctive normal form. Given a registered fragment of $R$, $F$, the question is: “if $p F$ is applicable to $F$, can we retrieve only $\sigma_{p F} F$ into the mediator and still evaluate $CSET(R, p, c)$ correctly?”

In order to decide whether we can push a predicate onto a fragment, we have to consider the impact of such reductions on the query semantics. Consider the fragments shown in Figure 3 and $C = CSET(Person, \text{Age} > 33)$. Assume we retrieve only $\sigma_{\text{Age} > 33}$ (Fragment 1) and $\sigma_{\text{Age} > 33}$ (Fragment 4) into the mediator. $t_{freq} = (003, \text{Fred}, 32, 3000)$ in Fragment 1 will not be retrieved. This potentially excludes 003 from $C$. If $c = \text{RandomEvidence}$, it is valid to exclude 003 from $C$, according to Definition 4.2. If $c = \text{HighConfidence}$ then it is necessary to exclude 003 from $C$. However, the mediator will retrieve $t_{freq} = (003, \text{Fred}, 34, 8, 7)$ from Fragment 4 and algorithm ComputeCSET would include 003 in $C$, resulting in an incorrect CSET. To fix this problem, we can send 003 to the site of Fragment 1 to verify that Fred indeed has Age $> 33$. In our example, the verification fails and 003 is removed from $C$. This process is referred to as PID verification. Obviously, when Age is supported by only one fragment, PID verification is not needed. Assume we have derived a temporary CSET value $C'$ from reduced fragments. To perform PID verification, we send the following queries to the sites of Fragment 1 and 4, respectively:

$$\delta_1 = C' \cap \pi_{PID} \sigma_{\text{Age} > 33} (\text{Fragment 1})$$

$$\delta_4 = C' \cap \pi_{PID} \sigma_{\text{Age} > 33} (\text{Fragment 4})$$

PID values in $\delta_1$ or $\delta_4$ must be removed from $C'$. The cost of this approach is low when (1) query selectivity is low resulting in a small $C'$, and (2) Conflict rate is low resulting in small $\delta$s. When no conflict exists, all $\delta$s are empty. When $C'$ is large, the cost of PID verification may offset the savings achieved by pushing selections onto fragments; a cost model is needed for strategy selection.

If $c = \text{PossibleAtAll}$, $C$ can be computed by ComputeCSET correctly from reduced fragments. However, we must be careful about pushing predicates that involve more than one attribute. Consider $C_1 = CSET(Person, \text{HomeNo}=\text{WorkNo}, \text{PossibleAtAll})$. In Figure 3, Fragment 2 contains tuple (003, Fred, 3003, 3000). If we retrieve only $\sigma_{\text{HomeNo}=\text{WorkNo}} (\text{Fragment 2})$, 003 will be excluded from $C_1$, which is incorrect since combining Fragments 1 and 2, it is possible that Fred’s HomeNo and WorkNo are the same, 3000. Generally, we can push a multi-attribute predicate $p$ onto a fragment $F$ only if no fragments other than $F$ support any of the attributes involved in $p$.

CT query optimization possibilities as illustrated by the example above are summarized in Table 3. In the next sec-
tion, we formally establish the above described optimization strategies. When \( c = \text{HighConfidence} \), a cost model is needed to determine whether the strategies we devise actually reduce cost. This is a future research issue; we only establish the validity of the strategies in this paper.

<table>
<thead>
<tr>
<th>( c )</th>
<th>Can ( p_x ) be used for fragment reduction?</th>
</tr>
</thead>
<tbody>
<tr>
<td>RandomEvidence</td>
<td>YES (RandomEvidence)</td>
</tr>
<tr>
<td>PossibleAtAll</td>
<td>YES (PossibleAtAll)</td>
</tr>
<tr>
<td>HighConfidence</td>
<td>YES (HighConfidence)</td>
</tr>
</tbody>
</table>

**Table 3. Fragment Reduction with Selections**

### 6.2. A Theory for CT query Optimization

The main theorems of our theory are Theorems 6.1 and 6.2, which allow us to push selections across MJ onto fragments to various degrees according to the PE-parameter.

**Theorem 6.1** Let \( R \) be a CA-relation. Let \( p = p_1 \land p_2 \land ... \land p_x \) be a predicate over \( R \) in conjunctive normal form. Let \( F_1, ..., F_n \) be all fragments registered with \( R \) that contain no null values. Let \( \bar{p} = p_1 \land ... \land p_x \), where \( p_x \in \{p_1, ..., p_x\} \). If \( 1 \leq j \leq s \) is applicable to \( F_i \).

Let \( F_i = \sigma_{p_{j1}}(F_i_i), 1 \leq i \leq n \). Let \( W = T_1 \{ \text{PID} \} \cup ... \cup T_n \{ \text{PID} \} \). Let \( R' = MJ(\text{PID}(R), \text{ATTR}(R), F_1, ..., F_n) \). Then we have the following:

1. \( \text{CSET}(R', p, \text{RandomEvidence}) \) is a valid value for \( \text{CSET}(R, p, \text{RandomEvidence}) \).
2. \( \text{CSET}(R, p, \text{HighConfidence}) = \text{CSET}(R', p, \text{HighConfidence}) \).

Note that 2 of Theorem 6.1 says that \( \text{CSET}(R, p, \text{HighConfidence}) \) can be computed from reduced fragments but we must verify that PID values thus selected are not in any \( T_x \). This process is the PID verification as described earlier.

**Theorem 6.2** Let \( R \) be a CA-relation. Let \( p = p_1 \land p_2 \land ... \land p_x \) be a predicate over \( R \) in conjunctive normal form. Let \( F_1, ..., F_n \) be all fragments registered with \( R \). \( F_i \)’s do not contain null values. Then

\[
\text{CSET}(R, p, \text{PossibleAtAll}) = \text{CSET}(MJ(R, F_1, ..., F_n), p, \text{PossibleAtAll})
\]

where \( \forall i, 1 \leq i \leq n, F_i = \sigma_{p_{ji}}(F_i_i), j \neq 1 \land ... \land j \neq s, p_x \in \{p_1, ..., p_x\} \).

1. \( \text{ATTR}(\bar{p}) = \{\text{PID}\} \) or \( \text{ATTR}(\bar{p}) = \text{ATTR}(p) \cap \text{ATTR}(F_i_i) \) and

2. \( p_j \notin \{1 \leq j \leq s \} \) involves at most one non-PID attribute or no registered fragment of \( R \) other than \( F_i \) supports any of the non-PID attributes in \( \text{ATTR}(\bar{p}) \).

We omit the formal proof of these theorems due to limit in space. Interested readers can find these proofs in [9].

### 6.3. Optimized CT Query Evaluation

The following algorithm is directly based on Theorems 6.1 and 6.2.

**Algorithm Optimized-CT-QP** \((R, Q, F_1, ..., F_n)\)

**input:**

- \( R \): Global relation \( R \) involved in the query.
- \( Q \): \( Q_{CT} = \text{select } I \text{ from } R \text{ where } p \text{ with } c_1 \).
- \( F_i \): All the fragments registered with \( R \).

**output:** \( A \): the query answer.

**begin**

**Compute CSET:**

- Let \( L_1 = \text{ATTR}(L) \cup \{\text{PID}(R)\} \cup \text{ATTR}(p) \). Write \( p \) into conjunctive normal form \( p = p_1 \land ... \land p_x \). Let \( X_p = \{p_1, ..., p_x\} \). For \( i = 1, n \) do:

- if \( c_1 \neq \text{PossibleAtAll} \) then let \( p_i \) be the conjunction of all predicates in \( X_p \) that are applicable to \( F_i \). If no such \( p_i \) is found, \( p_i = \text{true} \).
- if \( c_1 = \text{PossibleAtAll} \) then let \( p_i \) be the conjunction of all predicates in \( X_p \) such that (1) it involves at most one non-PID attribute; or (2) No fragments other than \( F_i \) supports any of the non-PID attributes involved. If \( \text{ATTR}(p_i) \neq \text{ATTR}(p) \cap \text{ATTR}(F_i) \), \( p_i = \text{true} \).

**S1** \( F_i = \pi_{L_1 \cap \text{ATTR}(F_i)} \sigma_{p_i}(F_i) \).

- \( R' = MJ(\text{PID}(R), L_1, F_1, ..., F_n) \);

- \( C = \text{ComputeCSET}(R', p, c_1) \).

**PID Verification:**

- If \( c_1 = \text{HighConfidence} \) or \( \text{DE}(L) \neq \text{ANY} \) then

  - Let \( L_2 = \text{ATTR}(L) \cup \{\text{PID}(R)\} \).

  - For \( i = 1, n \) do:

    **S2** Let \( \delta_i = \pi_{L_2 \cap \text{ATTR}(F_i)} \sigma_{p_i}(F_i) \). \( F_i \vdash \text{PID}(R) \). \( \delta_i \in C \).

- \( R' = R' \vdash \text{PID}(R) C \).

**Data Completion:**

- If \( \text{DE}(L) \neq \text{ANY} \) then \( R' \vdash \text{PID}(R) \).

**Data Extraction:**

- \( A = \pi_{\text{ATTR}(L)}[RTC( R' \vdash \text{PID}(R) C, \text{DE}(L))] \).
end of algorithm.

Steps S1 in Compute CSET and S2 in PID Verification are where queries are sent to the data sources that provide the respective fragments. These steps follow directly from Theorems 6.1 and 6.2. When the number of sources involved is large and data volume is large, cutting down on data retrieval at S1 and S2 improves query performance. We further observe the following:

Optimized-CT-QP is a 1- or 2-phase algorithm. The first phase retrieves enough data to compute CSET, depending on the PE- and the DE-parameter, a second phase retrieves extra data for PID verification and/or data completion. PID verification is only needed if the PE-parameter is HighConfidence. Data completion is not needed when the DE-parameter is ANY.

Performance perspectives of Optimized-CT-QP. Step S1 is obviously a good move towards saving communication cost. At step S2, we could send the content of the computed CSET, C, to relevant data sources. This works well when C is small due to a low query selectivity, but may get expensive when C is large. A simple computation can be applied to restrict this cost. Consider performing step S2 against a data source supporting a fragment $F_i$. The purpose of sending a query to compute $W_i$ is indeed to retrieve data related to PIDs in $F_i$ that are in C but have not been retrieved in step S1. Thus we can compare the volume of $\pi_{\text{PE-Attr}}(F_i)$ with the volume of C. If the former is smaller, then we simply retrieve it without sending C to the relevant data source, and proceed normally.

Optimized-CT-QP performs better when conflict rate is low. When conflict rate is low, the $\delta_i$'s will be empty or very small. This means the cost of PID verification and Data Completion becomes low. We expect Optimize-CT-QP to be most efficient against low conflict data.

**Example 6.1** We use the above given algorithms to evaluate the following 6 queries:

$$
\begin{align*}
\text{select } d \text{ Name, Age} \\
\text{from Person} \\
\text{where} \text{ Age} > 33 \text{ and HomeNo} = \text{WorkNo} \\
\text{and NoWorkYear} \geq \text{NoSchoolYear} \\
\text{with } c_1 \\
\text{where } c_1 \in \{\text{RandomEvidence, HighConfidence, PossibleAtAll}\} \text{ and } d \in \{\text{ANY, DISCARD}\}.
\end{align*}
$$

Figure 7 shows the predicates pushed onto $F_i$'s to compute $F_i$ (i = 1,...,4) in the case of HighConfidence and RandomEvidence. It also shows the result of the match join producing $R'$, from which we get: $CSET(Person, p, RandomEvidence) = \{004,005\}$. Based on this result we perform PID verification. The $\delta_i$'s are computed when $c_1 = \text{HighConfidence}$ or $d = \text{DISCARD}$, shown in Figure 9. Based on this result we have: $CSET(Person, p, HighConfidence) = \{005\}$ Figure 8 shows the predicates pushed onto $F_i$'s to compute of $F_i$'s, i = 1,...,4 in the case of $c_1 = \text{PossibleAtAll}$. It also shows the result of the match join producing $R'$, from which we get: $CSET(Person, p, PossibleAtAll) = \{003,004,005\}$ Final results of the 6 queries are given in Figure 10.
7. Related Work

Projects that employ a similar style data integration model include DISCO [7] and Information Manifold (IM) [4]. AURORA differs from these systems in its 2-tiered mediation model, designed to make adding and removing data sources easier. Unlike previous systems, we do not force the adoption of a specific data model; above the wrapper level, relational and object-oriented mediators can be developed independently. A comparison between AURORA and other mediation models can be found in [8, 9]. Many integration systems with comparable model do not deal with instance level conflicts.

[1, 2] study algebraic rules for pushing selections across aggregation functions, under the assumption that schema integration is performed by an integration specification which resolves all potential instance level conflicts using various aggregation functions. AURORA integration mediators do not keep integration specifications, sources participate in the data service by registering with the mediator the data they can contribute. Conflicts are not resolved at schema integration time but rather tolerated at query time and resolved only upon returning of query results. In general, AURORA’s approach towards instance level conflict handling offers a new way of querying potentially inconsistent data and new techniques for processing such queries efficiently.

The flexible relation model [3, 6] is designed to deal with instance level conflicts but it requires the applications to use a non-standard data model for data access. This approach only deals with conflicts at predicate evaluation time and the tolerance mode is always HighConfidence. Conflicts in query results are not removed. Multiplex [5] deals with instance level conflicts in the context of answering queries using given materialized views. Conflicts arise when the materialized views overlap and the same query can be evaluated in multiple ways resulting in multiple answers.
A mechanism is proposed to derive an approximate query answer using these candidate answers. However, without any object matching assumption, it is not clear how conflicts can be detected.

8. Conclusion and Future Work

In this paper, we described AURORA’s approach to instance level conflict handling. This approach differs from previous approaches in that we do not resolve conflicts at schema integration time with aggregation functions, we define a new model for querying possibly inconsistent data, the CT query model. With this model, conflicts are tolerated to a degree specified by the user at query time. The advantage of the CT query approach is that applications gain more control of the quality of the data access service they receive and the mediator gain more room for query optimization; we have developed techniques for optimized processing of such queries. We believe that the ability of optimizing query processing according to applications’ requirements is a significant factor in deployment.

Future research involves development of a cost model for strategy selection and a detailed performance study of the query optimization techniques presented here. Since query processing is a multi-phase procedure, apart from the major transformations given in this paper, many smaller techniques for smart reuse of data retrieved in previous phases can be explored. These are engineering issues but may improve performance further.

References


