

# Reconciling while Tolerating Disagreement in Collaborative Data Sharing

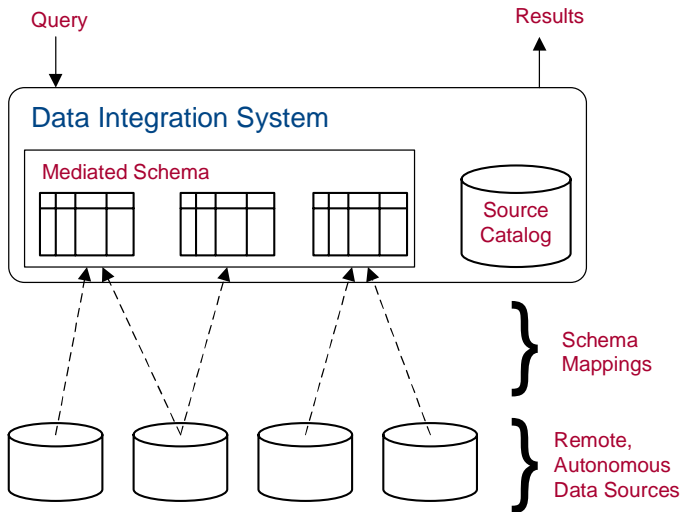
Nicholas Taylor, Zachary Ives

Department of Computer and Information Science



ACM SIGMOD  
International Conference on Management of Data  
June 27, 2006

- ▶ You keep the address books on your cell phone and PDA in sync. . .
  - ▶ only they contain slightly different data.
- ▶ Collaborators often share citation databases. . .
  - ▶ but prefer different abbreviation styles,
  - ▶ and use different citation programs.
- ▶ Biologists must keep their databases of protein functions up to date. . .
  - ▶ only they have disagreements about certain organisms.



- ▶ Sources are **independent**, and may have conflicting data.
  - ▶ Common in collaborative settings, such as the sciences.
  - ▶ Not studied in traditional data integration.
- ▶ Conflicts are revealed by schema constraints.
  - ▶ Each instance must satisfy the constraints individually.
  - ▶ However, together they may not.

## Conflicting Instances

<i>Address</i>	
<i>name</i>	<i>address</i>
Geno's	1219 S 9th
Pat's	1237 E Passyunk
Jim's	400 South
Steve's	7200 Bustleton

<i>Address</i>	
<i>name</i>	<i>address</i>
Geno's	1219 S 9th
Pat's	1237 E Passyunk
Jim's	431 N 62nd

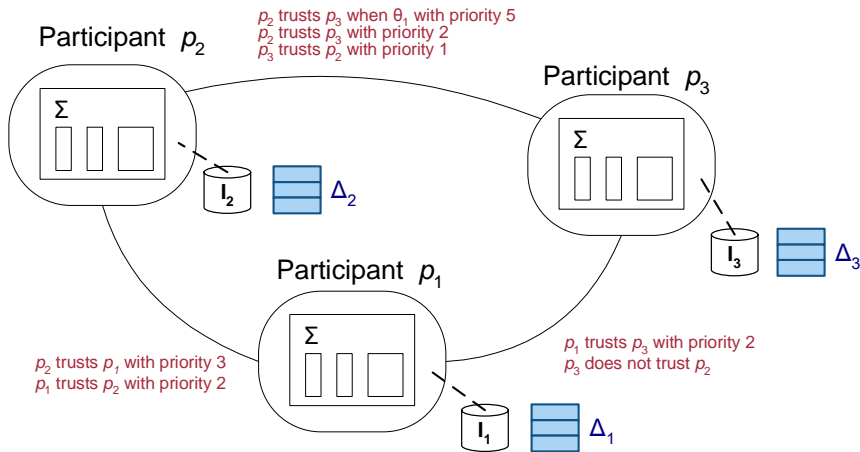
## Disclaimer

From here on we assume a single schema for the entire system. Other work in our group looks at translation.

## Disclaimer

From here on we assume a single schema for the entire system. Other work in our group looks at translation.

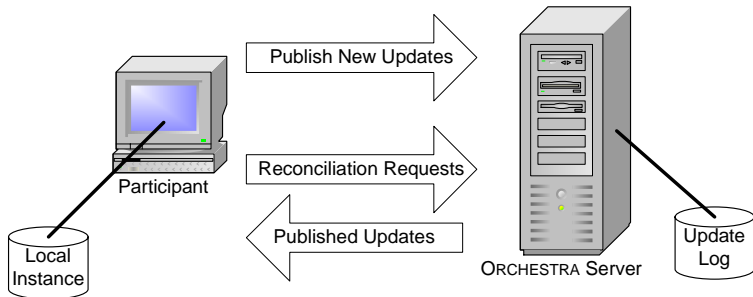
- ▶ Creating a unified data instance is impossible since sources may disagree.
- ▶ Instead, record the **updates** that each source makes.
- ▶ Updates are grouped into atomic transactions,
- ▶ Source choose which transactions to apply to its local instance via a user-specified policy, creating its own 'global instance.'



- ▶ Transaction priority is a composition of constituent tuple priorities.

- ▶ Intermittent connection to the system is tolerated.
  - ▶ Maximal progress is made at each step.
- ▶ System behavior is intuitive.
  - ▶ All transaction acceptances are final.
  - ▶ High priority transactions are always preferred.
- ▶ Reconciliation requires no user intervention.
  - ▶ Conflict resolution can be put off indefinitely.
  - ▶ User inattention has only temporary effects.

- ▶ Data sharing operations involve only one peer.
- ▶ Unlike CVS, publishing can never fail.



- ▶ Reconciling participant applies a **consistent** subset of updates it receives.

## Reconciliation 1

**+(A,4)** ✓

+(A,2) ✗

**+(D,8)** ⌚

**+(D,9)** ⌚

## Reconciliation 2

**+(A,3)** ✗

+(B,3) ✓  
+(C,5)

**+(C,6)** ✗

Txn Priority:

**High**

**Medium**

**Low**

**(B,3) → (B,4)** ✓



Accept



Reject



Defer

- ▶ The updates not modify non-existent values, or cause constraint violations.
- ▶ Before applying a transaction, we must have applied all antecedent transactions.
- ▶ After **flattening**, only one update is applied to any given value.
- ▶ Therefore when determining a consistent subset, we look at flattened **chains** of antecedent transactions.

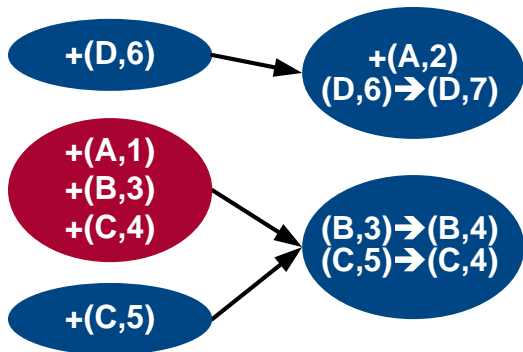
## Flattening

Flattening combines updates to remove transient conflicts, *e.g.*  $[+(A, 1), (A, 1) \rightarrow (B, 1)]$  becomes  $[+(B, 1)]$  This does not conflict with  $[+(A, 2)]$ .

The following greedy algorithm maximizes the number of high-priority transactions accepted (see paper for details).

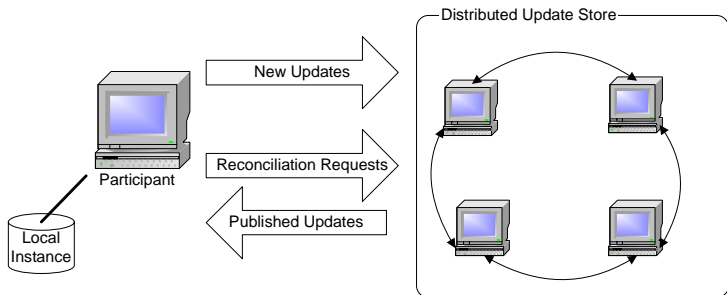
1. Let this reconciliation's set of accepted transactions  $A = \emptyset$ .
2. Flatten the antecedent chain for each trusted transaction (even if it is the antecedent of another one).
3. Reject inapplicable transaction chains.
4. Find all conflicts between transactions chains.
5. For each transaction priority  $p$ , in decreasing order:
  - 5.1 Let  $T$  be the set of trusted transactions with priority  $p$ .
  - 5.2 If a  $t \in T$  conflicts with a non-subsumed  $t' \in A$ , reject it.
  - 5.3 If a  $t \in T$  conflicts with a non-subsumed, non-rejected  $t' \in T$ , defer it.
  - 5.4 If a  $t \in T$  uses a dirty value, defer it.
  - 5.5 Accept all other  $t \in T$  by adding them to  $A$ .

- ▶ Suppose peer 1 is reconciling.
- ▶ It trusts both **peer 2** and **peer 3**, but prefers **peer 2**.
- ▶ All instances are initially empty.



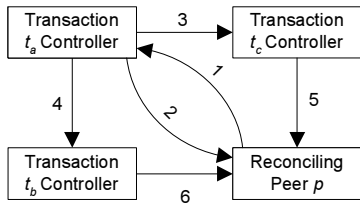
- ▶ Peer 1 accepts all but the top right transaction from **peer 3**.

- ▶ Reconciliation algorithm is implemented in Java.
- ▶ Two implementations of the **update store** hold published transactions.
  - ▶ DB2 used for centralized storage.
  - ▶ FreePastry used for distributed storage.



Distributed store also spreads some computation across network.

Suppose transaction  $t_a$  is trusted by reconciling peer  $p$ .  $t_a$  has two antecedents,  $t_b$  and  $t_c$ .  $t_c$  has no antecedents, and  $p$  has already accepted  $t_b$ 's antecedents.

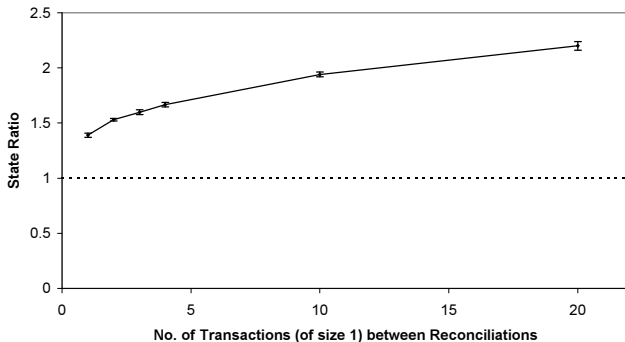


1.  $p$  requests  $t_a$
2.  $t_a$  controller sends  $t_a$  to  $p$
3.  $t_a$  controller requests  $t_c$  for  $p$
4.  $t_a$  controller requests  $t_b$  for  $p$
5.  $t_c$  controller sends  $t_c$  to  $p$
6.  $t_b$  controller sends  $t_b$  to  $p$

Now  $p$  has everything it needs to apply  $t_a$ .

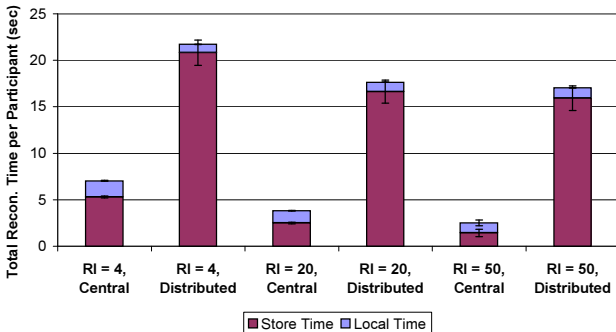
- ▶ Synthetic workload generator based on Swiss-Prot, data is skewed (heavy-tailed Zipfian with  $s = 1.5$ ).
- ▶ All transactions are of size one.
- ▶ All experiments involve ten peers.
- ▶ Experiments compare central and distributed update store.
  - ▶ Central update store and clients connected by 100Mb Ethernet.
  - ▶ Distributed update stores connected by simulated network with delay at least  $500 \mu\text{sec}$ .

Effect of reconciliation frequency on average **state ratio** for ten participants.



This shows that infrequent reconciliation slowly decreases the synchronicity between participants.



Total reconciliation times for ten peers publishing 500 transactions each.



This shows that infrequent reconciliation is more efficient, but only noticeably so for the centralized store. In the distributed store, network latency dominates.

- ▶ Infrequent reconciliation has only a slight impact on synchronicity and performance.
- ▶ See paper for details:
  - ▶ Transactions of size 2 cause decreased synchronicity, but after that the effect is negligible.
  - ▶ Adding peers decreases synchronicity sublinearly.
  - ▶ Processing time scales approximately linearly with the number of updates.

- ▶ Traditional data integration techniques **cannot tolerate conflicting data**.
- ▶ We propose ORCHESTRA as a **collaborative data sharing system**
  - ▶ with peer-centric semantics for consistency, and
  - ▶ a fully distributed architecture.
- ▶ Performance evaluations shows such a system is **feasible**.
  
- ▶ Future Work
  - ▶ Improved performance and reliability
  - ▶ Support for multiple schemas

 **ORCHESTRA** and  **HARMONY**  
solve two different problems.

- ▶ Harmony uses bidirectional translations to exactly synchronize data between schemas.
- ▶ ORCHESTRA extends traditional database mapping to translate between schemas, while allowing local control and disagreement.