

Reconciliation of Shared Data in ORCHESTRA

Nicholas Taylor, Zachary Ives

Department of Computer and Information Science
University of Pennsylvania

Systems Lunch
May 12, 2006

- ▶ You'd like to keep the address books on your cell phone and your PDA in sync. . .
 - ▶ only they contain slightly different data.

- ▶ You'd like to keep the address books on your cell phone and your PDA in sync. . .
 - ▶ only they contain slightly different data.
- ▶ Prof. A and Prof. B would like to share their citation databases entries. . .
 - ▶ only they prefer different abbreviation styles,
 - ▶ and Prof. A uses bibtex while Prof. B uses EndNote.

- ▶ You'd like to keep the address books on your cell phone and your PDA in sync. . .
 - ▶ only they contain slightly different data.
- ▶ Prof. A and Prof. B would like to share their citation databases entries. . .
 - ▶ only they prefer different abbreviation styles,
 - ▶ and Prof. A uses bibtex while Prof. B uses EndNote.
- ▶ Biologists C, D, and E would like to keep their databases of protein functions up to date. . .
 - ▶ only they have disagreements about certain organisms.

- ▶ Different data formats
- ▶ Different data fields at each source
- ▶ Different, contradictory data

- ▶ Different data formats
- ▶ Different data fields at each source
- ▶ Different, contradictory data

What does synchronization mean in this context?

Traditionally, translation between formats and fields is solved (to varying degrees of success) using **data integration** techniques.

Definition

Data integration is the problem of combining data residing at different sources and providing the user with a unified view of these data. (Lenzerini, 2002)

Traditionally, translation between formats and fields is solved (to varying degrees of success) using **data integration** techniques.

Definition

Data integration is the problem of combining data residing at different sources and providing the user with a unified view of these data. (Lenzerini, 2002)

We want to do something similar, but where **each** database becomes the 'unified view.'

- ▶ This talk discusses integration and sharing of relational databases.
- ▶ A relational database consists of
 - ▶ A schema Σ
 - ▶ An instance I of the relations in Σ

- ▶ This talk discusses integration and sharing of relational databases.
- ▶ A relational database consists of
 - ▶ A schema Σ
 - ▶ An instance I of the relations in Σ

Example database

$\Sigma = \text{Address}(\underline{\textit{name}}, \textit{address})$

	<i>name</i>	<i>Address</i> <i>address</i>
$I =$	Geno's	1219 S 9th
	Pat's	1237 E Passyunk
	Jim's	400 South

The sources may be **heterogeneous**, i.e. have different schemas.

Different schemas

$\Sigma = \text{Address}(\underline{\text{name}}, \text{address})$

$\Sigma' = \text{Name}(\underline{\text{rid}}, \text{name}), \text{Address}'(\underline{\text{rid}}, \text{address})$

The sources may be **heterogeneous**, i.e. have different schemas.

- ▶ Common in business settings as **Enterprise Information Integration**.

Different schemas

$$\Sigma = \text{Address}(\underline{\textit{name}}, \textit{address})$$
$$\Sigma' = \text{Name}(\underline{\textit{rid}}, \textit{name}), \text{Address}'(\underline{\textit{rid}}, \textit{address})$$

The sources may be **heterogeneous**, i.e. have different schemas.

- ▶ Common in business settings as **Enterprise Information Integration**.
- ▶ Translation between schemas using **mappings** is well studied.

Different schemas

$$\Sigma = \text{Address}(\underline{\textit{name}}, \textit{address})$$
$$\Sigma' = \text{Name}(\underline{\textit{rid}}, \textit{name}), \text{Address}'(\underline{\textit{rid}}, \textit{address})$$

Mappings

$$\forall n, a \text{ Address}(n, a) \Rightarrow \exists \textit{rid} \text{ Name}(\textit{rid}, n), \text{Address}'(\textit{rid}, a)$$
$$\forall \textit{rid}, n, a \text{ Name}(\textit{rid}, n), \text{Address}'(\textit{rid}, a) \Rightarrow \text{Address}(n, a)$$

- ▶ Since the sources are **independent**, they may have conflicting data.

- ▶ Since the sources are **independent**, they may have conflicting data.
 - ▶ Common in collaborative settings, such as the sciences.
 - ▶ Not studied in traditional data integration, and the focus of our work.

- ▶ Since the sources are **independent**, they may have conflicting data.
 - ▶ Common in collaborative settings, such as the sciences.
 - ▶ Not studied in traditional data integration, and the focus of our work.
- ▶ Conflicts arise in the presence of schema constraints.

Conflicting Instances

<i>Address</i>		<i>Address</i>	
<i>name</i>	<i>address</i>	<i>name</i>	<i>address</i>
Geno's	1219 S 9th	Geno's	1219 S 9th
Pat's	1237 E Passyunk	Pat's	1237 E Passyunk
Jim's	400 South	Jim's	431 N 62nd
Steve's	7200 Bustleton		

- ▶ Since the sources are **independent**, they may have conflicting data.
 - ▶ Common in collaborative settings, such as the sciences.
 - ▶ Not studied in traditional data integration, and the focus of our work.
- ▶ Conflicts arise in the presence of schema constraints.
 - ▶ Each instance must satisfy the constraints.
 - ▶ However, together they may not.

Conflicting Instances

<i>Address</i>		<i>Address</i>	
<i>name</i>	<i>address</i>	<i>name</i>	<i>address</i>
Geno's	1219 S 9th	Geno's	1219 S 9th
Pat's	1237 E Passyunk	Pat's	1237 E Passyunk
Jim's	400 South	Jim's	431 N 62nd
Steve's	7200 Bustleton		

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.

- ▶ A unified view of different sources is impossible since they may disagree.

Disclaimer

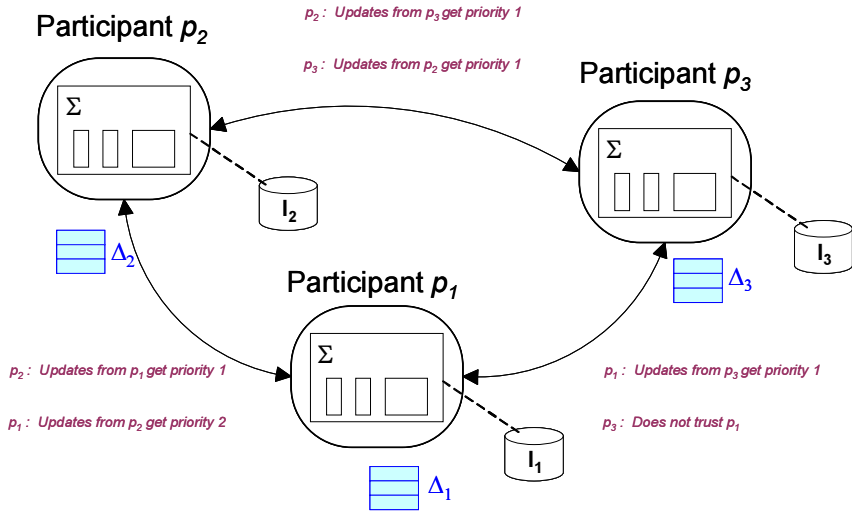
From here on we assume a single schema for the entire system. Other work in our group looks at translation.

- ▶ A unified view of different sources is impossible since they may disagree.
- ▶ Instead, record the **updates** that sources make and send them to other sources.

Disclaimer

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

- ▶ A unified view of different sources is impossible since they may disagree.
- ▶ Instead, record the **updates** that sources make and send them to other sources.
- ▶ Each source chooses which updates to apply to its local instance via a user-specified policy.



- ▶ All data sharing operations involve only one peer.

- ▶ All data sharing operations involve only one peer.
 - ▶ **Publishing** stores new updates to the local instance in a system-wide data structure.
 - ▶ **Reconciliation** retrieves updates published since the peer's last reconciliation.

- ▶ All data sharing operations involve only one peer.
 - ▶ **Publishing** stores new updates to the local instance in a system-wide data structure.
 - ▶ **Reconciliation** retrieves updates published since the peer's last reconciliation.
- ▶ This tolerates intermittent connection to the system.

- ▶ All data sharing operations involve only one peer.
 - ▶ **Publishing** stores new updates to the local instance in a system-wide data structure.
 - ▶ **Reconciliation** retrieves updates published since the peer's last reconciliation.
- ▶ This tolerates intermittent connection to the system.
- ▶ Note that, unlike CVS, publishing can never fail, since conflict resolution takes place at each instance.

- ▶ Updates may depend on **antecedent** updates.

- ▶ Updates may depend on **antecedent** updates.

Peer 1	Peer 2
+ (White Dog, 3422 Sansom)	
publish	
	reconcile (White Dog, 3422 Sansom) → (White Dog, 3420 Sansom)

- ▶ Updates may depend on **antecedent** updates.

Peer 1	Peer 2
+ (White Dog, 3422 Sansom)	
publish	
	reconcile (White Dog, 3422 Sansom) → (White Dog, 3420 Sansom)

- ▶ Here, update from Peer 2 depends on update from Peer 1.

- ▶ Updates are grouped into atomic **transactions**.
- ▶ Only **fully trusted** transactions are considered.

- ▶ Updates are grouped into atomic **transactions**.
- ▶ Only **fully trusted** transactions are considered.
- ▶ Suppose Peer 1 trusts Peer 2 for restaurants on Walnut Street.

- ▶ Updates are grouped into atomic **transactions**.
- ▶ Only **fully trusted** transactions are considered.
- ▶ Suppose Peer 1 trusts Peer 2 for restaurants on Walnut Street.

Fully trusted transactions

Peer 1 trusts Peer 2's transaction $\{+(Così Rittenhouse, 1623 Walnut)\}$.

- ▶ Updates are grouped into atomic **transactions**.
- ▶ Only **fully trusted** transactions are considered.
- ▶ Suppose Peer 1 trusts Peer 2 for restaurants on Walnut Street.

Fully trusted transactions

Peer 1 trusts Peer 2's transaction $\{+(Così Rittenhouse, 1623 Walnut)\}$.

Peer 1 does not trust Peer 2's transaction

$\{+(Così 12th, 1128 Walnut), +(Così UPenn, 140 S 36th)\}$

- ▶ Updates are retrieved through **reconciliation**, which brings in many new updates simultaneously.
- ▶ **Priorities** are used to resolve conflicts during a reconciliation, if possible.
- ▶ In a tie, both transactions are **deferred** until a human decides which to accept. Anything affected by this choice must also be deferred.

- ▶ Updates are retrieved through **reconciliation**, which brings in many new updates simultaneously.
- ▶ **Priorities** are used to resolve conflicts during a reconciliation, if possible.
- ▶ In a tie, both transactions are **deferred** until a human decides which to accept. Anything affected by this choice must also be deferred.

Conflicting transactions

Transactions $\{+(Così, 1128\ Walnut)\}$ and $\{+(Così, 140\ S\ 36th)\}$ conflict, since they contain updates that insert the same key with different values.

- ▶ The system should allow disconnected operation.

- ▶ The system should allow disconnected operation.
- ▶ Participants publish updates to the system periodically.

- ▶ The system should allow disconnected operation.
- ▶ Participants publish updates to the system periodically.
- ▶ The system may store them in
 - ▶ A central store (such as a database), or
 - ▶ A distributed store (such as a DHT).

- ▶ The system should allow disconnected operation.
- ▶ Participants publish updates to the system periodically.
- ▶ The system may store them in
 - ▶ A central store (such as a database), or
 - ▶ A distributed store (such as a DHT).
- ▶ Participants reconcile periodically, retrieving many updates.

- ▶ The system should allow disconnected operation.
- ▶ Participants publish updates to the system periodically.
- ▶ The system may store them in
 - ▶ A central store (such as a database), or
 - ▶ A distributed store (such as a DHT).
- ▶ Participants reconcile periodically, retrieving many updates.
- ▶ The participant then must choose a **subset** of the retrieved updates that
 - ▶ is consistent with its current state, and
 - ▶ does not conflict with itself.

- ▶ What does **consistency** mean?

- ▶ What does **consistency** mean?
 - ▶ The updates not modify non-existent values, or cause constraint violations.
 - ▶ After **flattening**, only one update is applied to any given value.

Flattening

Flattening combines updates to remove intermediate state. For example, $[+A, A \rightarrow B]$ becomes $[+B]$.

- ▶ What does **consistency** mean?
 - ▶ The updates not modify non-existent values, or cause constraint violations.
 - ▶ After **flattening**, only one update is applied to any given value.
- ▶ Before applying a transaction, we must have applied all antecedent transactions.

Flattening

Flattening combines updates to remove intermediate state. For example, $[+A, A \rightarrow B]$ becomes $[+B]$.

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

Flattening

Flattening combines updates to remove intermediate state. For example, $[+A, A \rightarrow B]$ becomes $[+B]$.

- ▶ For a particular reconciliation, there are many consistent subsets of new transactions.

- ▶ For a particular reconciliation, there are many consistent subsets of new transactions.
 - ▶ We assign each transaction the priority of its highest priority update.
 - ▶ If there is a conflict, we always prefer higher-priority transactions to lower-priority ones. . .

- ▶ For a particular reconciliation, there are many consistent subsets of new transactions.
 - ▶ We assign each transaction the priority of its highest priority update.
 - ▶ If there is a conflict, we always prefer higher-priority transactions to lower-priority ones. . .
 - ▶ . . . unless the lower-priority one **subsumes** the higher-priority one.

- ▶ For a particular reconciliation, there are many consistent subsets of new transactions.
 - ▶ We assign each transaction the priority of its highest priority update.
 - ▶ If there is a conflict, we always prefer higher-priority transactions to lower-priority ones. . .
 - ▶ . . . unless the lower-priority one **subsumes** the higher-priority one.
- ▶ This very naturally gives rise to a greedy algorithm for reconciliation.

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

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

Peer 2	Peer 3
<ul style="list-style-type: none">+(Geno's, 1219 S 9th)+(Steve's, 402 South) <p>publish</p>	<ul style="list-style-type: none">+(Geno's, 1218 S 9th)
	<p>reconcile</p> <ul style="list-style-type: none">(Steve's, 402 South) →(Steve's, 400 South) <p>publish</p>

- ▶ Peer 1 accepts...

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

Peer 2	Peer 3
+ (Geno's, 1219 S 9th) + (Steve's, 402 South) publish	+ (Geno's, 1218 S 9th)
	reconcile (Steve's, 402 South) → (Steve's, 400 South) publish

- ▶ Peer 1 accepts. . . all transactions except the first from peer 3.

- ▶ Suppose peer 1 is reconciling. It trusts peer 2 and peer 3 equally.
- ▶ All instances are initially empty.

- ▶ Suppose peer 1 is reconciling. It trusts peer 2 and peer 3 equally.
- ▶ All instances are initially empty.
- ▶ Relevant transactions:

Peer 2	Peer 3
+ (Geno's, 1219 S 9th) + (Steve's, 400 South) publish	+ (Geno's, 1218 S 9th) (Geno's, 1218 S 9th) → (Geno's, 1219 S 9th) + (Steve's, 402 South)
	publish

- ▶ Peer 1 accepts...

- ▶ Suppose peer 1 is reconciling. It trusts peer 2 and peer 3 equally.
- ▶ All instances are initially empty.
- ▶ Relevant transactions:

Peer 2	Peer 3
+ (Geno's, 1219 S 9th) + (Steve's, 400 South)	+ (Geno's, 1218 S 9th) (Geno's, 1218 S 9th) → (Geno's, 1219 S 9th) + (Steve's, 402 South)
publish	publish

- ▶ Peer 1 accepts... the first three because together they are compatible.
- ▶ The last two are deferred, and the key “Steve’s” is added to the **dirty set**.

- ▶ Reconciliation algorithm is implemented in Java.
- ▶ Two implementations of the **update stores** to hold published transactions

- ▶ Reconciliation algorithm is implemented in Java.
- ▶ Two implementations of the **update stores** to hold published transactions
 - ▶ Relational Database (DB2)

- ▶ Reconciliation algorithm is implemented in Java.
- ▶ Two implementations of the **update stores** to hold published transactions
 - ▶ Relational Database (DB2)
 - ▶ Distributed Hash Table (FreePastry)

DB2-backed store required extensive tuning to get reasonable performance.

DB2-backed store required extensive tuning to get reasonable performance.

- ▶ SQL constraints aren't expressive enough to let DBMS know about logs
 - ▶ Therefore automatic view maintenance (i.e. precomputation) is impossible.
 - ▶ We had to manually maintain result caches.

DB2-backed store required extensive tuning to get reasonable performance.

- ▶ SQL constraints aren't expressive enough to let DBMS know about logs
 - ▶ Therefore automatic view maintenance (i.e. precomputation) is impossible.
 - ▶ We had to manually maintain result caches.
- ▶ Recursive SQL performance was not good.
 - ▶ Client-side fixed-point recursion gave orders of magnitude performance improvement.

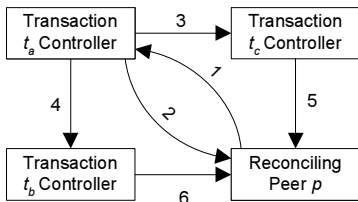
- ▶ Distributes all storage across peers
 - ▶ Published transactions
 - ▶ Accepted and rejected transactions

- ▶ Distributes all storage across peers
 - ▶ Published transactions
 - ▶ Accepted and rejected transactions
- ▶ Distributed computation across peers
 - ▶ Computation of antecedent transaction chains

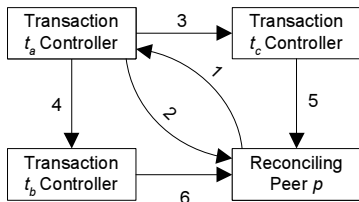
Suppose transaction t_a is trusted by reconciling peer p . t_a has two antecedents, t_b and t_c . t_c has no antecedents, and p has already accepted t_b 's antecedents.

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

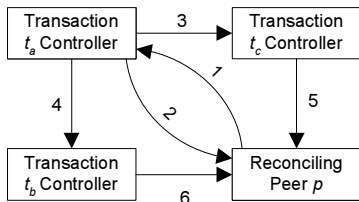


Suppose transaction t_a is trusted by reconciling peer p . t_a has two antecedents, t_b and t_c . t_c has no antecedents, and p has already accepted t_b 's antecedents.



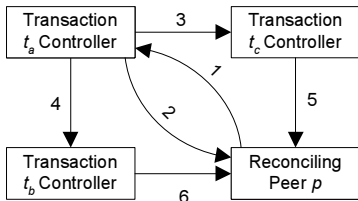
1. p requests t_a
2. t_a controller sends t_a to p

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.



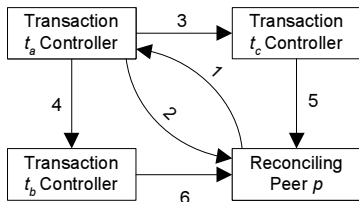
1. p requests t_a
2. t_a controller sends t_a to p
3. t_a controller requests t_c for p

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.



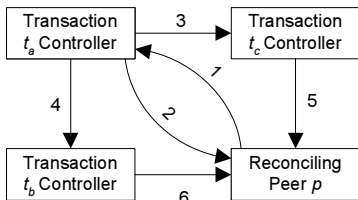
1. p requests t_a
2. t_a controller sends t_a to p
3. t_a controller requests t_c for p
4. t_a controller requests t_b for p

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.



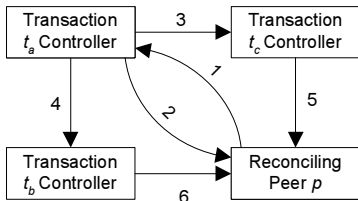
1. p requests t_a
2. t_a controller sends t_a to p
3. t_a controller requests t_c for p
4. t_a controller requests t_b for p
5. t_c controller sends t_c to p

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

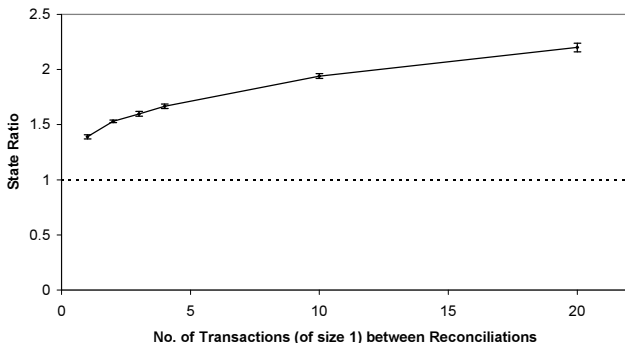
Suppose transaction t_a is trusted by reconciling peer p . t_a has two antecedents, t_b and t_c . t_c has no antecedents, and p has already accepted t_b 's antecedents.



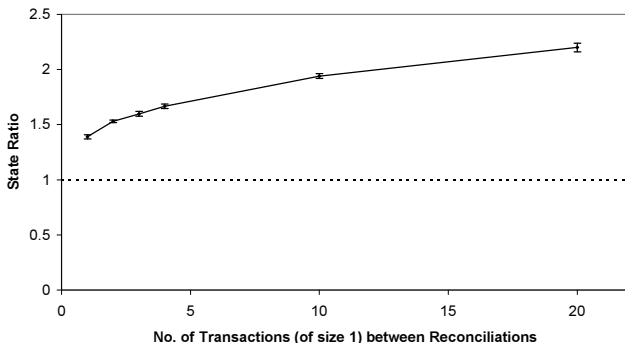
1. p requests t_a
2. t_a controller sends t_a to p
3. t_a controller requests t_c for p
4. t_a controller requests t_b for p
5. t_c controller sends t_c to p
6. t_b controller sends t_b to p

Now p has everything it needs to apply t_a .

Here we show the effect of reconciliation frequency on average **state ratio** for ten participants.

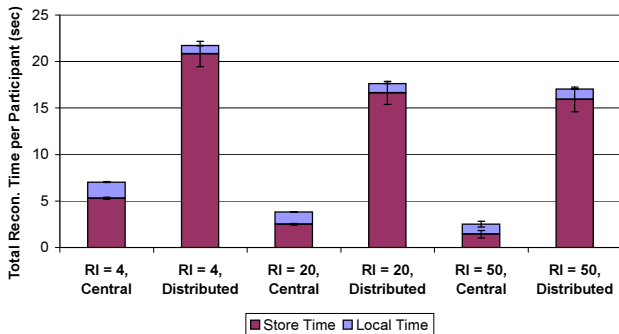


Here we show the effect of reconciliation frequency on average **state ratio** for ten participants.

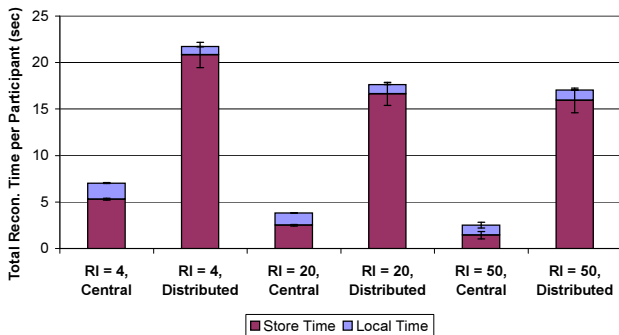


This shows that infrequent reconciliation slowly decreases the synchronicity between participants.

Here we show the total reconciliation times for ten peers publishing 500 transactions each.



Here we show the 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.

- ▶ Traditional data integration techniques **cannot tolerate conflicting data**.
- ▶ We propose ORCHESTRA as a **collaborative data sharing system** that can.
- ▶ Performance evaluations shows such a system is **feasible**.



- ▶ Future Work
 - ▶ Improved performance and reliability
 - ▶ Support for multiple schemas (ask TJ and Greg for details!)
 - ▶ Direct integration with RDBMS

 **ORCHESTRA** and
solve two different problems.

 **HARMONY**

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

- ▶ Harmony uses bidirectional translations to exactly synchronize data between 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.