Exploiting Schemas in Data Synchronization

Nate Foster (Penn)
Michael B. Greenwald (Lucent)
Christian Kirkegaard (BRICS)
Benjamin C. Pierce (Penn)
Alan Schmitt (INRIA)

HARMONY

Optimistic Replication

- Many copies of distributed data stored on often disconnected hosts
- Any copy may be updated at any time
- Hosts occasionally ***synchronize***
  - Merging updates that they agree on
  - Resolving conflicting updates

Optimistic Replication

- Many copies of distributed data stored on often disconnected hosts
- Any copy may be updated at any time
- Hosts occasionally ***synchronize***
  - Merging updates that they agree on
  - Resolving conflicting updates
- Many advantages: availability, scalability, quality control
- Main challenge: synchronization

> "...based on the ***optimistic presumption*** that ***conflicting updates are rare***, and that the ***contents are consistent enough*** with those on other replicas." —Saito & Shapiro (2002)

HARMONY Project

Research goal: Facilitate optimistic replication by building a generic synchronization framework for heterogeneous, tree-structured data.

This talk: Focus on Harmony's synchronization algorithm.

- ***Local:*** intuitive, easy to predict behavior
- ***Schema-aware***: preserves structural invariants

Running Example

XML Address Book

```xml
<xcard>
  <vcard>
    <n>Steve</n>
    <org>Stanford</org>
    <email>freunds@cs.stanford.edu</email>
  </vcard>
  <vcard>
    <n>Kim</n>
    <org>Williams</org>
    <email>kim@cs.williams.edu</email>
  </vcard>
</xcard>
```

Updated Address Book

```
<xcard>
  <vcard>
    <n>Steve</n>
    <org>Williams</org>
    <email>freund@cs.williams.edu</email>
  </vcard>
  <vcard>
    <n>Kim</n>
    <org>Williams</org>
    <email>kim@cs.williams.edu</email>
  </vcard>
</xcard>
```

Another Update

```
<xcard>
  <vcard>
    <n>Kim</n>
    <org>Pomona</org>
    <email>kim@cs.pomona.edu</email>
  </vcard>
  <vcard>
    <n>Steve</n>
    <org>Stanford</org>
    <email>freunds@cs.stanford.edu</email>
  </vcard>
</xcard>
```

Goal: Synchronized Address Book

```
<xcard>
  <vcard>
    <n>Steve</n>
    <org>Williams</org>
    <email>freund@cs.williams.edu</email>
  </vcard>
  <vcard>
    <n>Kim</n>
    <org>Pomona</org>
    <email>kim@cs.pomona.edu</email>
  </vcard>
</xcard>
```

Data Model

Trees

Harmony's data model is unordered, edge-labeled trees where every child of a node has a distinct name.

Equivalently, a tree is a partial function from names to trees.

$$\left\{ \begin{array}{r} \texttt{email} \mapsto \left\{ \texttt{kim@cs.williams.edu} \mapsto \{\} \right\} \\ \texttt{n} \mapsto \left\{ \texttt{Kim} \mapsto \{\} \right\} \\ \texttt{org} \mapsto \left\{ \texttt{Williams} \mapsto \{\} \right\} \end{array} \right\}$$

Trees

Harmony's data model is unordered, edge-labeled trees where every child of a node has a distinct name.

Equivalently, a tree is a partial function from names to trees.

$$\left\{ \begin{array}{r} \texttt{email} \mapsto \left\{ \texttt{kim@cs.williams.edu} \mapsto \{\} \right\} \\ \texttt{n} \mapsto \left\{ \texttt{Kim} \mapsto \{\} \right\} \\ \texttt{org} \mapsto \left\{ \texttt{Williams} \mapsto \{\} \right\} \end{array} \right\}$$

Within a tree, we'll abbreviate $\texttt{k} \mapsto \{\}$ as $\texttt{k}$.

Lists

Lists are encoded as "cons cells"; the list

$$\left[t_1, t_2, \ldots, t_n\right]$$

is represented by

$$\left\{\begin{array}{l} \texttt{hd} \mapsto t_1 \\ \texttt{tl} \mapsto \left\{\begin{array}{l} \texttt{hd} \mapsto t_2 \\ \texttt{tl} \mapsto \left\{ \ldots \mapsto \left\{\begin{array}{l} \texttt{hd} \mapsto t_n \\ \texttt{tl} \mapsto \left\{\texttt{nil}\right\} \end{array}\right\}\right\} \end{array}\right\} \end{array}\right\}$$

XML

The XML element

```
<tag>
   subelt_1 ... subelt_n
</tag>
```

is represented by the tree

$$\left\{ \texttt{tag} \mapsto \begin{bmatrix} \langle \texttt{subelt}_1 \rangle \\ \vdots \\ \langle \texttt{subelt}_n \rangle \end{bmatrix} \right\}$$

Encoded Address Book

The original XML address book, encoded as a tree:

$$\left\{ \texttt{xcard} \mapsto \left[\begin{array}{l} \texttt{vcard} \mapsto \left[\begin{array}{r} \texttt{n} \mapsto \left[\texttt{Steve} \mapsto [] \right] \\ \texttt{org} \mapsto \left[\texttt{Stanford} \mapsto [] \right] \\ \texttt{email} \mapsto \left[\texttt{freunds@cs.stanford.edu} \mapsto [] \right] \end{array} \right] \\ \texttt{vcard} \mapsto \left[\begin{array}{r} \texttt{n} \mapsto \left[\texttt{Kim} \mapsto [] \right] \\ \texttt{org} \mapsto \left[\texttt{Williams} \mapsto [] \right] \\ \texttt{email} \mapsto \left[\texttt{kim@cs.williams.edu} \mapsto [] \right] \end{array} \right] \end{array} \right] \right\}$$

Synchronization: Simple Algorithm

Notation

- ***names***, ranged over by $\mathtt{k}$
- a ***path*** p is a sequence of names
- a ***tree*** is a finite map from names to trees
- the ***contents*** of a tree t at some name k, written $t(k)$, is either a tree or $\bot$
- write $\mathcal{T}$ for the set of all trees
- write $\mathcal{T}_\bot = \mathcal{T} \cup \{\bot\}$
- $\mathcal{X}$ is a special tree that marks conflicts in the archive

Simple Algorithm

$$sync \quad \in \quad (\mathcal{T}_{\mathcal{X}\perp} \times \mathcal{T}_{\perp} \times \mathcal{T}_{\perp}) \longrightarrow (\mathcal{T}_{\mathcal{X}\perp} \times \mathcal{T}_{\perp} \times \mathcal{T}_{\perp})$$

$sync(o, a, b) =$

if $a = b$ then (a, a, b) – *equal replicas: done*

else if $a = o$ then (b, b, b) – *no change to* a

else if $b = o$ then (a, a, a) – *no change to* b

else if $o = \mathcal{X}$ then (o, a, b) – *unresolved conflict*

else if $a = \perp$ then $(\mathcal{X}, a, b)$ – *delete/modify conflict*

else if $b = \perp$ then $(\mathcal{X}, a, b)$ – *delete/modify conflict*

else – *proceed recursively...*

let $(o'(k), a'(k), b'(k)) = sync(o(k), a(k), b(k))$

$\forall k \in dom(a) \cup dom(b)$ in

(o', a', b')

Uh oh...

Two problems: (1) entries are not aligned correctly; (2) synchronizer doesn't preserve schemas!

$$\left\{ \texttt{xcard} \mapsto \begin{bmatrix} \texttt{vcard} \mapsto \begin{bmatrix} \texttt{n} \mapsto \left[\texttt{Kim} \mapsto [] \right] \\ \texttt{org} \mapsto \begin{bmatrix} \texttt{Pomona} \mapsto [] \\ \texttt{Williams} \mapsto [] \end{bmatrix} \\ \texttt{email} \mapsto \begin{bmatrix} \texttt{freunds@cs.williams.edu} \mapsto [] \\ \texttt{kim@cs.pomona.edu} \mapsto [] \end{bmatrix} \end{bmatrix} \\ \texttt{vcard} \mapsto \begin{bmatrix} \texttt{n} \mapsto \left[\texttt{steve} \mapsto [] \right] \\ \texttt{org} \mapsto \left[\texttt{Stanford} \mapsto [] \right] \\ \texttt{email} \mapsto \left[\texttt{freunds@cs.stanford.edu} \mapsto [] \right] \end{bmatrix} \end{bmatrix} \right\}$$

Alignment and Lenses

Alignment

Alignment consists of identifying the parts of each replica that represent the "same data".

Alignment

Alignment consists of identifying the parts of each replica that represent the "same data". Two approaches:

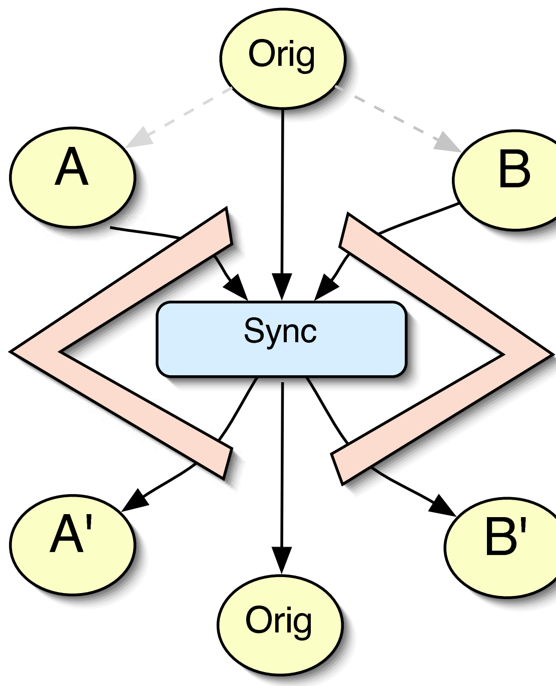
- ***Global*** alignment strategies analyze the entire replica to come up with a "best alignment". Usually heuristic (e.g., minimizing "edit distance").
  Examples: Diff-based tools.
- ***Local*** alignment strategies are simpler; e.g., align the the children with the same name.
  - To be effective, we must pre-align the replicas so that the common structure is exposed.

Lenses

- Can pre-align replicas by transforming them ***before*** synchronization.
  - E.g., for XML address books encoded as trees, can discard order and lift up a key from each entry.
- After synchronization, we must “undo” the transform to recover the original format.
- Harmony includes a domain-specific language for writing bi-directional transformations on trees, called ***lenses***.
  - Every well-typed program is “well-behaved”.
- (Also facilitates ***heterogeneous*** data synchronization.)

Synchronization Architecture

Each replica is passed through a lens before and after synchronization.



Pre-aligning with lenses

$$\left\{ \texttt{xcard} \mapsto \left[\begin{array}{l} \texttt{vcard} \mapsto \left[\begin{array}{rl} \texttt{n} \mapsto & \left[\texttt{Steve} \mapsto [\,] \right] \\ \texttt{org} \mapsto & \left[\texttt{Stanford} \mapsto [\,] \right] \\ \texttt{email} \mapsto & \left[\texttt{freunds@cs.stanford.edu} \mapsto [\,] \right] \end{array} \right] \\ \texttt{vcard} \mapsto \left[\begin{array}{rl} \texttt{n} \mapsto & \left[\texttt{Kim} \mapsto [\,] \right] \\ \texttt{org} \mapsto & \left[\texttt{Williams} \mapsto [\,] \right] \\ \texttt{email} \mapsto & \left[\texttt{kim@cs.williams.edu} \mapsto [\,] \right] \end{array} \right] \end{array} \right] \right\}$$

`hoist "xcard"`

Pre-aligning with lenses

```
[ vcard ↦ [ n ↦ [ Steve ↦ [] ]
            org ↦ [ Stanford ↦ [] ]
            email ↦ [ freunds@cs.stanford.edu ↦ [] ] ]
  vcard ↦ [ n ↦ [ Kim ↦ [] ]
            org ↦ [ Williams ↦ [] ]
            email ↦ [ kim@cs.williams.edu ↦ [] ] ] ]
```

```
hoist "xcard";
List.map (hoist "vcard")
```

Pre-aligning with lenses

$$\left[\begin{array}{l}\left[\begin{array}{rl}\texttt{n} \mapsto & [\texttt{Steve} \mapsto []] \\ \texttt{org} \mapsto & [\texttt{Stanford} \mapsto []] \\ \texttt{email} \mapsto & [\texttt{freunds@cs.stanford.edu} \mapsto []]\end{array}\right] \\ \left[\begin{array}{rl}\texttt{n} \mapsto & [\texttt{Kim} \mapsto []] \\ \texttt{org} \mapsto & [\texttt{Williams} \mapsto []] \\ \texttt{email} \mapsto & [\texttt{kim@cs.williams.edu} \mapsto []]\end{array}\right]\end{array}\right]$$

```
hoist "xcard";
List.map (hoist "vcard";
          List.flatten; map(List.hd []; List.hd []))
```

Pre-aligning with lenses

$$\left[\begin{array}{l}\left\{\begin{array}{r}\texttt{email}\mapsto\{\texttt{freunds@cs.stanford.edu}\mapsto[]\}\\ \texttt{n}\mapsto\{\texttt{Steve}\mapsto[]\}\\ \texttt{org}\mapsto\{\texttt{Stanford}\mapsto[]\}\end{array}\right\}\\ \left\{\begin{array}{r}\texttt{email}\mapsto\{\texttt{kim@cs.williams.edu}\mapsto[]\}\\ \texttt{n}\mapsto\{\texttt{Kim}\mapsto[]\}\\ \texttt{org}\mapsto\{\texttt{Williams}\mapsto[]\}\end{array}\right\}\end{array}\right]$$

```
hoist "xcard";
List.map (hoist "vcard";
          List.flatten; map(List.hd []; List.hd [];
                            map (const {} [])))
```

Pre-aligning with lenses

$$\left[\begin{array}{l}\left\{\begin{array}{r}\texttt{email}\mapsto\{\texttt{freunds@cs.stanford.edu}\}\\ \texttt{n}\mapsto\{\texttt{Steve}\}\\ \texttt{org}\mapsto\{\texttt{Stanford}\}\end{array}\right\}\\ \left\{\begin{array}{r}\texttt{email}\mapsto\{\texttt{kim@cs.williams.edu}\}\\ \texttt{n}\mapsto\{\texttt{Kim}\}\\ \texttt{org}\mapsto\{\texttt{Williams}\}\end{array}\right\}\end{array}\right]$$

```
hoist "xcard";
List.map (hoist "vcard";
          List.flatten; map(List.hd []; List.hd [];
                            map (const {} []));
          pivot "n")
```

Pre-aligning with lenses

$$\left[\begin{array}{r} \texttt{Steve} \mapsto \left\{\begin{array}{r} \texttt{email} \mapsto \{\texttt{freunds@cs.stanford.edu}\} \\ \texttt{org} \mapsto \{\texttt{Stanford}\} \end{array}\right\} \\ \texttt{Kim} \mapsto \left\{\begin{array}{r} \texttt{email} \mapsto \{\texttt{kim@cs.williams.edu}\} \\ \texttt{org} \mapsto \{\texttt{Williams}\} \end{array}\right\} \end{array}\right]$$

```
hoist "xcard";
List.map (hoist "vcard";
          List.flatten; map(List.hd []; List.hd [];
                                  map (const {} []));
          pivot "n");
List.flatten; map(List.hd [])
```

Pre-aligning with lenses

$$\left\{\begin{array}{r}\texttt{Kim}\mapsto\left\{\begin{array}{r}\texttt{email}\mapsto\{\texttt{kim@cs.williams.edu}\}\\ \texttt{org}\mapsto\{\texttt{Williams}\}\end{array}\right\}\\ \texttt{Steve}\mapsto\left\{\begin{array}{r}\texttt{email}\mapsto\{\texttt{freunds@cs.stanford.edu}\}\\ \texttt{org}\mapsto\{\texttt{Stanford}\}\end{array}\right\}\end{array}\right\}$$

```
hoist "xcard";
List.map (hoist "vcard";
          List.flatten; map(List.hd []; List.hd [];
                                map (const {} []));
          pivot "n");
List.flatten; map(List.hd [])
```

Schema-Aware Synchronization

Mangled Results

The synchronization algorithm is still a bit too eager: it will often merge changes in ways that yield mangled results.

$$o = \left\{ \texttt{org} \mapsto \left\{ \texttt{Williams} \right\} \right\}$$

$$a = \left\{ \texttt{org} \mapsto \left\{ \texttt{UC Santa Cruz} \right\} \right\}$$

$$b = \left\{ \texttt{org} \mapsto \left\{ \texttt{Pomona} \right\} \right\}$$

$$a' = b' = \left\{ \texttt{org} \mapsto \left\{ \begin{array}{l} \texttt{Pomona} \\ \texttt{UC Santa Cruz} \end{array} \right\} \right\}$$

More Difficulties

Similarly, suppose we want every address book entry to contain either an email address or an organization.

- start with a record containing both `email` and `org`
- delete `email` in one replica
- delete `org` in the other replica
- note that all three variants satisfy
- now synchronize...
- both deletions get propagated, yielding an ill-formed result.

A Simple Schema-Aware Synchronizer

$bettersync(S, o, a, b) =$

let $(o', a', b') = sync(o, a, b)$ in

if $(a' \notin S)$ or $(b' \notin S)$

then $(\mathcal{X}, a, b)$ – *schema conflict*

else (o', a', b')

A step too far...

This algorithm is too coarse-grained: A schema conflict ***anywhere*** results in a synchronization failure ***everywhere***!

We need to detect schema violations locally...

Final Algorithm

$sync(S, o, a, b) =$

if $a = b$ then (a, a, b) – *equal replicas: done*

else if $a = o$ then (b, b, b) – *no change to a*

else if $b = o$ then (a, a, a) – *no change to b*

else if $o = \mathcal{X}$ then (o, a, b) – *unresolved conflict*

else if $a = \perp$ then $(\mathcal{X}, a, b)$ – *delete/modify conflict*

else if $b = \perp$ then $(\mathcal{X}, a, b)$ – *delete/modify conflict*

else – *proceed recursively...*

let $(o'(k), a'(k), b'(k)) = sync(S(k), o(k), a(k), b(k))$

$\forall k \in dom(a) \cup dom(b)$ in

if $(dom(a') \notin doms(S))$ or $(dom(b') \notin doms(S))$

then $(\mathcal{X}, a, b)$ – *schema conflict*

else (o', a', b')

Path Consistency

To ensure that we can "project" a schema one a given name, we need to consider only schemas of a restricted form.

Definition: A schema S is ***path consistent*** iff, for all trees $t, t' \in S$ and paths p, we have

$$t(p) \neq \bot \ \wedge \ t'(p) \neq \bot \implies t[p \mapsto t'(p)] \in S,$$

where $t[p \mapsto t'(p)]$ is the tree obtained by replacing the subtree of t at p by the corresponding subtree of t'.

Path Consistency

To ensure that we can "project" a schema one a given name, we need to consider only schemas of a restricted form.

Definition: A schema S is ***path consistent*** iff, for all trees $t, t' \in S$ and paths p, we have

$$t(p) \neq \bot \ \wedge \ t'(p) \neq \bot \implies t[p \mapsto t'(p)] \in S,$$

where $t[p \mapsto t'(p)]$ is the tree obtained by replacing the subtree of t at p by the corresponding subtree of t'.

Path-consistent schemas are a "semantic analog" of ***single-type tree grammars*** used in W3C Schema. They are expressive enough to describe a wide range of examples.

Specification

A good synchronizer should...

1. Never "back out" changes
2. Never "make up" contents
3. Stop at conflicting paths (leaving replicas in their current states)
4. Always leave the replicas in a well-typed form

safety conditions

5. Propagate as many changes as possible without violating above rules

maximality condition

The (Theoretical) Punchline

Theorem: The final (schema-aware) synchronization algorithm is safe and maximal.

Proof: See paper.

The (Practical) Punchline

Bookmark Synchronizer Demo

Implementation Status

- Core implementation and several demos running:
  - bookmarks (Mozilla, Safari, Internet Explorer)
  - XML address books
  - structured text
- Unison integration coming soon.
- Public release this summer!

Acknowledgments

Collaborators on this work: Michael Greenwald, Christian Kirkegaard, Benjamin Pierce, and Alan Schmitt.

Other Harmony contributors: Malo Denielou, Owen Gunden, Sanjeev Khanna, Christian Kirkegaard, Keshav Kunal, Stéphane Lescuyer, Jonathan Moore, Thang Nguyen, and Zhe Yang.

http://www.cis.upenn.edu/∼bcpierce/harmony