GEAS: Generic Adaptive Scheduling for High-efficiency Context Inconsistency Detection

Bingying Guo, Huiyan Wang, Chang Xu, Jian Lu

State Key Laboratory for Novel Software Technology
Department of Computer Science and Technology
Nanjing University
Outline

1. Background and existing work
   - Context-aware applications
   - Constraint checking techniques

2. Problem
   - How to balance immediate scheduling and batch-based scheduling?

3. Our approach
   - Generic adaptive scheduling (GEAS)

4. Evaluation
   - Controlled experiments and real-world scenarios

5. Conclusion and future plan
Outline

1. Background and existing work
   - Context-aware applications
   - Constraint checking techniques

2. Problem
   - How to balance immediate scheduling and batch-based scheduling?

3. Our approach
   - Generic adaptive scheduling (GEAS)

4. Evaluation
   - Controlled experiments and real-world scenarios

5. Conclusion and future plan
Background (1)

1. Context-aware applications
   - Use contexts to understand environment
     - E.g., user location, room temperature, GPS data, etc.
   - Make adaptation based on contexts
     - E.g., a location-aware printer\(^7\), context-aware applications such as “X-10 (smartHome)”, “baiduMap”, “didi”, “mobike” and etc.
Background (1)

1. Context-aware applications
   - Use contexts to understand environment
     - E.g., user location, room temperature, GPS data, etc.
   - Make adaptation based on contexts
     - E.g., a location-aware printer[^7], context-aware applications such as “X-10 (smartHome)”, “baiduMap”, “didi”, “mobike” and etc.
Background (1)

1. Context-aware applications
   - Use *contexts* to understand environment
     - E.g., user location, room temperature, GPS data, etc.
   - Make adaptation based on contexts
     - E.g., a location-aware printer\cite{7}, context-aware applications such as “X-10 (smartHome)”, “baiduMap”, “didi”, “mobike” and etc.
Background (2)
Background (2)

1. Vulnerable to various noises
   - Context being inaccurate, incomplete or conflicting to each other[7]
     - RFID: over 30% missing and cross reads[1, 2, 3]
Background (2)

1. Vulnerable to various noises
   - Context being inaccurate, incomplete or conflicting to each other\cite{7}
     - RFID: over 30% missing and cross reads\cite{1, 2, 3}

How to validate contexts’ accuracy before actual usage?
Background (3)
1. How to validate contexts’ accuracy before actual usage?
   - No precise oracle to validate the accuracy directly
Background (3)

1. How to validate contexts’ accuracy before actual usage?
   - No precise oracle to validate the accuracy directly
   - One popular technique: *constraint checking*
     - Checking contexts against pre-specified *consistency constraints*\(^{[4, 10]}\) see whether any violation (named *context inconsistency*\(^{[6, 9]}\)) occurs
Background (3)

1. How to validate contexts’ accuracy before actual usage?
   - No precise oracle to validate the accuracy directly
   - One popular technique: constraint checking
     - Checking contexts against pre-specified consistency constraints\cite{4,10} see whether any violation (named context inconsistency\cite{6,9}) occurs
     - Consistency constraints: usually derived from general physical laws or application-specific requirements
       - “no one can stay in rooms x and y at the same time”
Background (3)

1. How to validate contexts’ accuracy before actual usage?
   - No precise oracle to validate the accuracy directly
   - One popular technique: constraint checking
     - Checking contexts against pre-specified consistency constraints\cite{4, 10} see whether any violation (named context inconsistency\cite{6, 9}) occurs

- Consistency constraints: usually derived from general physical laws or application-specific requirements
  - “no one can stay in rooms x and y at the same time”

\[
S_{loc} = \forall v_1 \in R_x \left( \neg \exists v_2 \in R_y \left( \text{equal}(v_1, v_2) \right) \right)
\]
Background (4)

1. Existing practice

❖ Detect context inconsistency more and more efficiently
   - Entire constraint checking [4] (ECC)
   - Partial constraint checking [6] (PCC)
   - Concurrent constraint checking [5] (Con-C)
   - GPU-based consistency checking [8] (GAIN)

❖ Immediate scheduling of constraint checking
   - Schedule constraint checking immediately upon each context change
Background (5)

1. Existing constraint checking techniques
1. Existing constraint checking techniques
Problem (1)

1. What if it comes to heavy-workload scenarios?
Problem (2)

1. What if it comes to heavy-workload scenarios?
Problem (3)

1. Existing techniques cannot handle heavy-workload scenarios.
   - Unfortunately, such scenarios are common in practice
Problem (3)

1. Existing techniques cannot handle heavy-workload scenarios.
   - Unfortunately, such scenarios are common in practice
   - Intuitively, sampling may be an option
     - Not schedule constraint checking upon collecting any context change
Problem (3)

1. Existing techniques cannot handle heavy-workload scenarios.
   ❖ Unfortunately, such scenarios are common in practice
   ❖ Intuitively, sampling may be an option
     • Not schedule constraint checking upon collecting any context change
     • Batch-based scheduling: check upon collecting fixed number of context changes

PCC/Con-C/GAIN
Problem (3)

1. Existing techniques cannot handle heavy-workload scenarios.
   - Unfortunately, such scenarios are common in practice
   - Intuitively, sampling may be an option
     • Not schedule constraint checking upon collecting any context change
     • Batch-based scheduling: check upon collecting fixed number of context changes

PCC/Con-C/GAIN
Problem (3)

1. Existing techniques cannot handle heavy-workload scenarios.
   - Unfortunately, such scenarios are common in practice
   - Intuitively, sampling may be an option
     - Not schedule constraint checking upon collecting any context change
     - Batch-based scheduling: check upon collecting fixed number of context changes

However, it may lead to inconsistency missing problem
Exhibition example (1)

Immediate Scheduling (ImmedSched)

- $p_1$ enters room $x$
- $p_1$ enters room $y$
- $p_3$ enters room $y$
- $p_3$ enters room $x$
- $p_2$ leaves room $x$

Batch-based Scheduling (BatchSched)

- $p_2$ enters room $x$
- $p_1$ leaves room $x$
- $p_2$ enters room $y$
- $p_3$ leaves room $y$

Inaccurate changes are indicated by $chg_1, chg_2, chg_3, chg_4, chg_5, chg_6, chg_7, chg_8, chg_9$.
Exhibition example (1)

1. Considered consistency constraint
   - “No one can stay in rooms x and y at the same time.”
   - $S_{loc} = \forall v_1 \in R_x \ (\neg \exists v_2 \in R_y (equal(v_1, v_2)))$

Immediate Scheduling (ImmedSched)

Batch-based Scheduling (BatchSched)
Exhibition example (1)

1. Considered consistency constraint
   - "No one can stay in rooms x and y at the same time."
   - \( S_{loc} = \forall v_1 \in R_x (\neg (\exists v_2 \in R_y (equal(v_1, v_2)))) \)

---

**Immediate Scheduling (ImmedSched)**
- \( p_1 \) enters room \( x \)
- \( p_1 \) enters room \( y \)
- \( p_3 \) enters room \( y \)
- \( p_3 \) enters room \( x \)
- \( p_2 \) leaves room \( x \)

**Batch-based Scheduling (BatchSched)**
- \( p_2 \) enters room \( x \)
- \( p_1 \) leaves room \( x \)
- \( p_2 \) enters room \( y \)
- \( p_3 \) leaves room \( y \)

**Inexact Data**

- \( chg_1 \)
- \( chg_2 \)
- \( chg_3 \)
- \( chg_4 \)
- \( chg_5 \)
- \( chg_6 \)
- \( chg_7 \)
- \( chg_8 \)
- \( chg_9 \)

---

1. **Constraint Checking Result** (CCR) is a set containing all context inconsistencies detected by certain constraint checking techniques. (snapshot)
2. **Inconsistency Detection Result** (IDR) is a union set of all obtained CCRs before. (accumulation)
Exhibition example (2)

1. Considered consistency constraint
   - “No one can stay in rooms x and y at the same time.”
   - \( S_{loc} = \forall v_1 \in R_x (\neg \exists v_2 \in R_y (equal(v_1, v_2))) \)

Immediate Scheduling (ImmedSched)

- \( p_1 \) enters room \( x \)
- \( p_1 \) enters room \( y \)
- \( p_3 \) enters room \( y \)
- \( p_3 \) enters room \( x \)
- \( p_2 \) leaves room \( x \)

\[ slc = \forall v_1 \in R_x (\neg \exists v_2 \in R_y (equal(v_1, v_2))) \]

<table>
<thead>
<tr>
<th>#</th>
<th>( p_1 )</th>
<th>( p_2 )</th>
<th>( p_3 )</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( p_1 )</td>
<td>( p_2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>( p_3 )</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>( p_2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( p_1 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>( p_2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>( p_3 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>( p_2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>( p_3 )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CCR:
- \( \{inc_1\} \)
- \( \{inc_2\} \)
- \( \{inc_1, inc_2\} \)
- \( \{inc_1, inc_2, inc_3\} \)

IDR:
- \( \{inc_1\} \)
- \( \{inc_1\} \)
- \( \{inc_1\} \)
- \( \{inc_1, inc_2\} \)
- \( \{inc_1, inc_2, inc_3\} \)
- \( \{inc_1\} \)
1. Considered consistency constraint

❖ "No one can stay in rooms x and y at the same time."

- $S_{loc} = \forall v_1 \in R_x (\neg \exists v_2 \in R_y (equal(v_1, v_2)))$

---

**Immediate Scheduling (ImmedSched)**

- $p_1$ enters room $x$
- $p_1$ enters room $y$
- $p_3$ enters room $y$
- $p_3$ enters room $x$
- $p_2$ leaves room $x$
- $p_2$ enters room $x$
- $p_1$ leaves room $x$
- $p_2$ enters room $y$
- $p_3$ leaves room $y$

---

**Ideal, and can detect all context inconsistencies**

- **ImmedSched**
- **CCR:** $\{\}$ $\{\}$ $\{inc_1\}$ $\{\}$ $\{\}$ $\{inc_2\}$ $\{inc_2, inc_3\}$ $\{inc_2\}$ $\{\}$
- **IDR:** $\{\}$ $\{\}$ $\{inc_1\}$ $\{inc_1\}$ $\{inc_1\}$ $\{inc_1, inc_2\}$ $\{inc_2, inc_3\}$ $\{inc_2, inc_3\}$ $\{inc_1, inc_2, inc_3\}$
Exhibition example (3)

1. Considered consistency constraint
   - "No one can stay in rooms x and y at the same time."
     - $S_{loc} = \forall v_1 \in R_x (\neg \exists v_2 \in R_y (equal(v_1, v_2)))$
1. Considered consistency constraint

❖ "No one can stay in rooms x and y at the same time."

• \( S_{loc} = \forall v_1 \in R_x \ (\text{not}(\exists v_2 \in R_y (equal(v_1, v_2)))) \)
1. Considered consistency constraint

❖ "No one can stay in rooms x and y at the same time."

\[ S_{loc} = \forall v_1 \in R_x (\neg \exists v_2 \in R_y (equal(v_1, v_2))) \]

lead to up to 66.7% inconsistency missing rate
Exhibition example (3)

Such missed inconsistencies can lead to app’s misbehaviors or even some severe consequences.

Batch-based Scheduling (BatchSched)

lead to up to 66.7% inconsistency missing rate

BatchSched \((\text{win.size} = 2)\)

CCR:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
<td>{inc2}</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
</tbody>
</table>

IDR:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
<td>{inc2}</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
</tbody>
</table>

\{inc1, inc2, inc3\}
Outline

1. Background and existing work
   - Context-aware applications
   - Constraint checking techniques

2. Problem
   - How to balance immediate scheduling and batch-based scheduling?

3. Our approach
   - Generic adaptive scheduling (GEAS)

4. Evaluation
   - Controlled experiments and real-world scenarios

5. Conclusion and future plan
How to balance?

1. Immediate scheduling
   - Ideal, but cannot handle heavy-workload scenarios

2. Batch-based scheduling
   - Efficient, but may cause inconsistency missing problem
Outline

1. Background and existing work
   ❖ Context-aware applications
   ❖ Constraint checking techniques

2. Problem
   ❖ How to balance immediate scheduling and batch-based scheduling?

3. Our approach
   ❖ Generic adaptive scheduling (GEAS)

4. Evaluation
   ❖ Controlled experiments and real-world scenarios

5. Conclusion and future plan
Our approach
Our approach

1. Aim to achieve
   - Efficient inconsistency detection by sampling
   - Complete inconsistency detection results
Our approach

1. Aim to achieve
   - Efficient inconsistency detection by sampling
   - Complete inconsistency detection results

2. Key observation
   - Only grouping certain combinations of context changes in a batch can cause missed inconsistencies in IDR
     - Suspicious pairs
Our approach

1. Aim to achieve
   - Efficient inconsistency detection by sampling
   - Complete inconsistency detection results

2. Key observation
   - Only grouping certain combinations of context changes in a batch can cause missed inconsistencies in IDR
     - Suspicious pairs
   - Avoiding arranging such suspicious pairs in a single batch, by adaptively tuning batch window
Overview
Overview

Consistency constraints → Deriving suspicious pairs from constraints → Suspicious pairs
Overview

Consistency constraints → Deriving suspicious pairs from constraints → Suspicious pairs

Matching changes against suspicious pairs

Stream of context changes
Overview

- Consistency constraints
- Deriving suspicious pairs from constraints
- Suspicious pairs
- Scheduling constraint checking
- Matching changes against suspicious pairs
- Scheduling decisions
- Stream of context changes
Overview

Consistency constraints → Deriving suspicious pairs from constraints → Suspicious pairs

Scheduling constraint checking ← Matching changes against suspicious pairs

Scheduling decisions

GEneric Adaptive Scheduling (GEAS)

Stream of context changes
Overview

Consistency constraints

Deriving suspicious pairs from constraints

Suspicious pairs

Scheduling constraint checking

Matching changes against suspicious pairs

Scheduling decisions

GEneric Adaptive Scheduling (GEAS)

Constraint checking techniques

Detected context inconsistencies

Stream of context changes
Deriving suspicious pair statically (1)

1. \(\text{inc}+/−/? \text{ change}\)
   - \(\text{inc}+ \text{ change}: \text{can only cause new inconsistencies}\)
   - \(\text{inc}− \text{ change}: \text{can only cause existing inconsistencies undetectable}\)
   - \(\text{inc}? \text{ change}: \text{can cause both}\)
Deriving suspicious pair statically (1)

1. inc+/−/? change
   - inc+ change: can only cause new inconsistencies
   - inc− change: can only cause existing inconsistencies undetectable
   - inc? change: can cause both

2. Suspicious pair
   - Combination of two context changes
     - First change (inc+ or inc?): can cause new inconsistencies
     - Second change (inc− or inc?): can cause existing context inconsistencies undetectable
Deriving suspicious pair statically (1)

1. inc+/−/? change
   - inc+ change: can only cause new inconsistencies
   - inc− change: can only cause existing inconsistencies undetectable
   - inc? change: can cause both

2. Suspicious pair
   - Combination of two context changes
     - First change (inc+ or inc?): can cause new inconsistencies
     - Second change (inc− or inc?): can cause existing context inconsistencies undetectable

Deriving suspicious pair statically (2)

1. Base rules for inc+/−/? changes
   - E.g., universal formula: \( \forall v \in C[f] \)
     - *addition change*: inc+ change

   **Reported inconsistencies:**
   - case 1: \( \emptyset \rightarrow \{(v, v_{k+1})\} \)
   - case 2: \( \{(v, v_2)\} \rightarrow \{(v, v_2), (v, v_{k+1})\} \)

   **case 1:**
   - \( v_1 \):
     - true
   - \( v_2 \):
     - true
   - \( v_{k-1} \):
     - true
   - \( v_k \):
     - true
   - Add \( v_{k+1} \) to \( C \)

   **case 2:**
   - \( v_1 \):
     - true
   - \( v_2 \):
     - false
   - \( v_{k-1} \):
     - true
   - \( v_k \):
     - true
   - Add \( v_{k+1} \) to \( C \)

   **false**
1. Base rules for inc+/−/? changes

- **E.g., universal formula:** $\forall v \in C[f]$
  - *deletion change*: inc− change

```
\text{Reported inconsistencies:}
\{(v, v_{k+1})\} \Rightarrow \emptyset
\{(v, v_2), (v, v_{k+1})\} \Rightarrow \{(v, v_2)\}
```

```
\forall v \in C[f]
\begin{array}{cccc}
  \ \ \ v_1 & v_2 & \ldots & v_{k-1} & v_k \\
  f & f & \ldots & f & f \\
\end{array}
```

- **case 1:** true  true  true  false  false
- **case 2:** true  false  true  false  false

Delete $v_k$ from $C$
Deriving suspicious pair statically (4)

1. Base rules for inc+/−/? changes
   - E.g., universal formula: $\forall v \in C[f]$
     - *update change*: inc? change

\[
\begin{align*}
\text{Reported inconsistencies:} & \quad \{(v, v_k)\} \rightarrow \emptyset \\
\emptyset & \rightarrow \{(v, v_k)\}
\end{align*}
\]

- **case 1:**
  - true true

- **case 2:**
  - true true

\[
\begin{align*}
\forall v \in C[f] & \\
\text{Update } v_k \text{ from } C
\end{align*}
\]

- **case 1:**
  - true false

- **case 2:**
  - true true
1. Deduction rules for inc+, inc− and inc? changes
   - Input: consistency constraints (FOL language)
   - Output: three sets of changes (inc+, inc− and inc?)

<table>
<thead>
<tr>
<th>Formula type</th>
<th>Deduction rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall v \in C \ (f)$</td>
<td>$\text{Set}_{inc^+}(f) \cup {&lt;+, C&gt;}$</td>
</tr>
<tr>
<td>$\exists v \in C \ (f)$</td>
<td>$\text{Set}_{inc^+}(f) \cup {&lt;-, C&gt;}$</td>
</tr>
<tr>
<td>$(f_1)$ and $(f_2)$</td>
<td>$\text{Set}<em>{inc^+}(f_1) \cup \text{Set}</em>{inc^+}(f_2)$</td>
</tr>
<tr>
<td>$(f_1)$ or $(f_2)$</td>
<td>$\text{Set}<em>{inc^+}(f_1) \cup \text{Set}</em>{inc^+}(f_2)$</td>
</tr>
<tr>
<td>$(f_1)$ implies $(f_2)$</td>
<td>$\text{Set}<em>{inc^-}(f_1) \cup \text{Set}</em>{inc^+}(f_2)$</td>
</tr>
<tr>
<td>not $(f)$</td>
<td>$\text{Set}_{inc^-}(f)$</td>
</tr>
<tr>
<td>$\text{bfunc}(v_1, ...)$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
Deriving suspicious pair statically (5)

1. Deduction rules for inc+, inc− and inc? changes

- Input: consistency constraints (FOL language)
- Output: three sets of changes (inc+, inc− and inc?)

<table>
<thead>
<tr>
<th>Formula type</th>
<th>Set of inc+ changes</th>
<th>Set of inc− changes</th>
<th>Set of inc? changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \forall v \in C(f) )</td>
<td>( Set_{inc+}(f) \cup {&lt;+, C&gt;} )</td>
<td>( Set_{inc-}(f) \cup {&lt;-, C&gt;} )</td>
<td>( Set_{inc?}(f) \cup {&lt;#, C&gt;} )</td>
</tr>
<tr>
<td>( \exists v \in C(f) )</td>
<td>( Set_{inc+}(f) \cup {&lt;-, C&gt;} )</td>
<td>( Set_{inc-}(f) \cup {&lt;+, C&gt;} )</td>
<td>( Set_{inc?}(f) \cup {&lt;#, C&gt;} )</td>
</tr>
<tr>
<td>(f₁) and (f₂)</td>
<td>( Set_{inc+}(f₁) \cup Set_{inc+}(f₂) )</td>
<td>( Set_{inc-}(f₁) \cup Set_{inc-}(f₂) )</td>
<td>( Set_{inc?}(f₁) \cup Set_{inc?}(f₂) )</td>
</tr>
<tr>
<td>(f₁) or (f₂)</td>
<td>( Set_{inc+}(f₁) \cup Set_{inc+}(f₂) )</td>
<td>( Set_{inc-}(f₁) \cup Set_{inc-}(f₂) )</td>
<td>( Set_{inc?}(f₁) \cup Set_{inc?}(f₂) )</td>
</tr>
<tr>
<td>(f₁) implies (f₂)</td>
<td>( Set_{inc-}(f₁) \cup Set_{inc+}(f₂) )</td>
<td>( Set_{inc+}(f₁) \cup Set_{inc-}(f₂) )</td>
<td>( Set_{inc?}(f₁) \cup Set_{inc?}(f₂) )</td>
</tr>
<tr>
<td>not (f)</td>
<td>( Set_{inc-}(f) )</td>
<td>( Set_{inc+}(f) )</td>
<td>( Set_{inc?}(f) )</td>
</tr>
<tr>
<td>bfunc(v₁,...)</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
Matching and scheduling (1)

1. Matching and scheduling
   - Matching such combination against suspicious pairs
     - The newly collected context change
     - Existing context changes in a batch
   - Scheduling constraint checking if matched successfully
     - Deal with all context changes in the current batch
Matching and scheduling (1)

1. Matching and scheduling
   - Matching such combination against suspicious pairs
     - The newly collected context change
     - Existing context changes in a batch
   - Scheduling constraint checking if matched successfully
     - Deal with all context changes in the current batch
Matching and scheduling (1)

1. Matching and scheduling
   - Matching such combination against suspicious pairs
     - The newly collected context change
     - Existing context changes in a batch
   - Scheduling constraint checking if matched successfully
     - Deal with all context changes in the current batch
Matching and scheduling (1)

1. Matching and scheduling
   - Matching such combination against suspicious pairs
     - The newly collected context change
     - Existing context changes in a batch
   - Scheduling constraint checking if matched successfully
     - Deal with all context changes in the current batch
Matching and scheduling

1. Matching and scheduling
   - Matching such combination against suspicious pairs
     - The newly collected context change
     - Existing context changes in a batch
   - Scheduling constraint checking if matched successfully
     - Deal with all context changes in the current batch
1. Matching and scheduling

- **Matching such combination against suspicious pairs**
  - The newly collected context change
  - Existing context changes in a batch

- **Scheduling constraint checking if matched successfully**
  - Deal with all context changes in the current batch
Matching and scheduling

1. Matching and scheduling
   - Matching such combination against suspicious pairs
     - The newly collected context change
     - Existing context changes in a batch
   - Scheduling constraint checking if matched successfully
     - Deal with all context changes in the current batch
Matching and scheduling

1. Matching and scheduling
   - Matching such combination against suspicious pairs
     - The newly collected context change
     - Existing context changes in a batch
   - Scheduling constraint checking if matched successfully
     - Deal with all context changes in the current batch
Overview

Consistency constraints

Deriving suspicious pairs from constraints

Suspicious pairs

Scheduling constraint checking

Matching changes against suspicious pairs

Scheduling decisions

GEneric Adaptive Scheduling (GEAS)

Constraint checking techniques

Detected context inconsistencies

Stream of context changes
Overview

Consistency constraints → Deriving suspicious pairs from constraints → Suspicious pairs

Scheduling constraint checking → Matching changes against suspicious pairs

Constraint checking techniques → Detected context inconsistencies

GEneric Adaptive Scheduling (GEAS)

Stream of context changes
Existing checking techniques (1)

1. Non-cache-based
   - Support processing multiple context changes together
     - do not depending on cached previous checking results
     - E.g., ECC\textsuperscript{[4]}, Con-C\textsuperscript{[5]}, GAIN\textsuperscript{[8]}

GEAS is directly applicable with non-cache-based checking techniques

2. Cache-based
   - Only support processing context changes one by one
     - depending on cached previous checking results
     - E.g., PCC\textsuperscript{[6]}

Need a little adjustment to apply GEAS with such cache-based checking techniques
Existing checking techniques (2)

1. Adjustment to PCC's semantics (named MPCC)

\[ T[\forall v \in C(f)]_\alpha = \]

1) \[ T_0[\forall v \in C(f)]_\alpha, \]
   if \( C \) has no change (i.e., \( C = C_0 \)) and \( \text{affected}(f) = \perp \);

2) \[ T_0[\forall v \in C(f)]_\alpha \wedge T[f]\text{bind}((v,x_1),\alpha) \wedge \ldots \wedge T[f]\text{bind}((v,x_n),\alpha) \mid x_i \in C - C_0, \]
   if \( C \) has addition changes only;

3) \[ T \wedge T_0[f]\text{bind}((v,x_1),\alpha) \wedge \ldots \wedge T_0[f]\text{bind}((v,x_m),\alpha) \wedge T[f]\text{bind}((v,y_1),\alpha) \wedge \ldots \wedge T[f]\text{bind}((v,y_n),\alpha) \mid x_i \in C_0 \cap C, \]
   if \( C \) has any deletion change (deletion changes only, or both addition and deletion changes);

4) \[ T \wedge T[f]\text{bind}((v,x_1),\alpha) \wedge \ldots \wedge T[f]\text{bind}((v,x_n),\alpha) \mid x_i \in C, \]
   if \( \text{affected}(f) = T \).

\[ L[\forall v \in C(f)]_\alpha = \]

1) \[ L_0[\forall v \in C(f)]_\alpha, \]
   if \( C \) has no change (i.e., \( C = C_0 \)) and \( \text{affected}(f) = \perp \);

2) \[ L_0[\forall v \in C(f)]_\alpha \cup \{ l \mid l \in \{(\text{violated}, \{(v, x_i)\})\} \otimes L[f]\text{bind}((v,x_i),\alpha) \mid x_i \in C - C_0 \wedge T[f]\text{bind}((v,x_i),\alpha) = \perp, \]
   if \( C \) has addition changes only;

3) \[ \{ l \mid l \in \{(\text{violated}, \{(v, x_i)\})\} \otimes L_0[f]\text{bind}((v,x_i),\alpha) \\} \cup \{ l \mid l \in \{(\text{violated}, \{(v, y_i)\})\} \otimes L[f]\text{bind}((v,y_i),\alpha) \mid x_i \in C_0 \cap C \]
   \wedge T[f]\text{bind}((v,x_i),\alpha) = \perp, y_i \in C - C_0 \wedge T[f]\text{bind}((v,y_i),\alpha) = \perp, \]
   if \( C \) has any deletion change (deletion changes only, or both addition changes and deletion changes);

4) \[ \{ l \mid l \in \{(\text{violated}, \{(v, x_i)\})\} \otimes L[f]\text{bind}((v,x_i),\alpha) \mid x_i \in C \wedge T[f]\text{bind}((v,x_i),\alpha) = \perp, \]
   if \( \text{affected}(f) = T \).
Existing checking techniques (2)

1. Adjustment to PCC’s semantics (named MPCC)

\[ \tau[\forall v \in C(f)]_\alpha = \]
\[ \begin{align*}
1) \quad & \tau_0[\forall v \in C(f)]_\alpha, \\
& \text{if } C \text{ has no change (i.e., } C = C_0 \text{) and } \text{affected}(f) = \bot; \\
2) \quad & \text{CCR equivalence and efficiency equivalence}
\end{align*} \]

**Theorem 1 (CCR equivalence):** Given any consistency constraint \( s \) and context pool \( P \), MPCC always returns the same CCR(\( s, P \)) value as PCC does.

**Theorem 2 (Efficiency equivalence):** Given any consistency constraint and context pool, MPCC does not increase the time complexity for constraint checking.
GEAS’ usage

1. Usage for the exhibition example

\[ p_1 \text{ enters room } x \quad p_1 \text{ enters room } y \quad p_3 \text{ enters room } y \quad p_3 \text{ enters room } x \quad p_2 \text{ leaves room } x \]

\[ p_2 \text{ enters room } x \quad p_1 \text{ leaves room } x \quad p_2 \text{ enters room } y \quad p_3 \text{ leaves room } y \]

<table>
<thead>
<tr>
<th>chg_1</th>
<th>chg_2</th>
<th>chg_3</th>
<th>chg_4</th>
<th>chg_5</th>
<th>chg_6</th>
<th>chg_7</th>
<th>chg_8</th>
<th>chg_9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Inaccurate

<table>
<thead>
<tr>
<th>ImmedSched</th>
<th>CCR:</th>
<th>IDR:</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>{}</td>
<td>{inc_1}</td>
<td>{inc_1}</td>
</tr>
<tr>
<td>{}</td>
<td>{}</td>
<td>{inc_1}</td>
</tr>
<tr>
<td>{}</td>
<td>{inc_2}</td>
<td>{inc_1, inc_2}</td>
</tr>
<tr>
<td>{inc_2}</td>
<td>{inc_2, inc_3}</td>
<td>{inc_1, inc_2, inc_3}</td>
</tr>
<tr>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
</tbody>
</table>

Time
GEAS’ usage

1. Usage for the exhibition example

$p_1$ enters room $x$  $p_1$ enters room $y$  $p_3$ enters room $y$  $p_3$ enters room $x$  $p_2$ leaves room $x$

$p_2$ enters room $x$  $p_1$ leaves room $x$  $p_2$ enters room $y$  $p_3$ leaves room $y$

Inaccurate

Derived 16 suspicious pairs:

$(<+, R_x/R_y>, <-, R_x/R_y>)$

$(<+, R_x/R_y>, <#, R_x/R_y>)$

$(<#, R_x/R_y>, <-, R_x/R_y>)$

$(<#, R_x/R_y>, <#, R_x/R_y>)$
GEAS’ usage

1. Usage for the exhibition example

Derived 16 suspicious pairs:

\[(<+, R_x/R_y>, <-, R_x/R_y>)\]
\[(<+, R_x/R_y>, <#, R_x/R_y>)\]
\[(<#, R_x/R_y>, <-, R_x/R_y>)\]
\[(<#, R_x/R_y>, <#, R_x/R_y>)\]
GEAS’ usage

1. Usage for the exhibition example

- \( p_1 \) enters room \( x \)
- \( p_1 \) enters room \( y \)
- \( p_3 \) enters room \( y \)
- \( p_3 \) enters room \( x \)
- \( p_2 \) leaves room \( x \)

GEAS

<table>
<thead>
<tr>
<th>Time</th>
<th>( chg_1 )</th>
<th>( chg_2 )</th>
<th>( chg_3 )</th>
<th>( chg_4 )</th>
<th>( chg_5 )</th>
<th>( chg_6 )</th>
<th>( chg_7 )</th>
<th>( chg_8 )</th>
<th>( chg_9 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( p_1 ) enters</td>
<td>( p_1 ) enters</td>
<td>( p_3 ) enters</td>
<td>( p_3 ) enters</td>
<td>( p_2 ) leaves</td>
<td>( p_3 ) leaves</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x )</td>
<td>( y )</td>
<td>( y )</td>
<td>( x )</td>
<td>( x )</td>
<td>( y )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Inaccurate

- \( chg_3 \)
- \( chg_4 \)
- \( chg_6 \)
- \( chg_7 \)

GEAS

| CCR: | IDR: |
|------|------|------|------|------|------|
| \{inc\_1\} | \{inc\_1\} | | | | |
| | | | | | |

Derived 16 suspicious pairs:

- \((<+, R_x/R_y>, <-, R_x/R_y>)\)
- \((<+, R_x/R_y>, <#, R_x/R_y>)\)
- \((<#, R_x/R_y>, <-, R_x/R_y>)\)
- \((<#, R_x/R_y>, <#, R_x/R_y>)\)
GEAS’ usage

1. Usage for the exhibition example

- Derived 16 suspicious pairs:
  - ($<+, Rx/Ry>$, $<-, Rx/Ry>$)
  - ($<+, Rx/Ry>$, $#Rx/Ry>$)
  - ($#Rx/Ry>$, $<-, Rx/Ry>$)
  - ($#Rx/Ry>$, $#Rx/Ry>$)

- GEAS
  - CCR: $\{inc_1\}$
  - IDR: $\{inc_1\}$
  - ImmedSched: $\{inc_1, inc_2, inc_3\}$

- CCR: $\{inc_1\}$ $\{inc_2\}$ $\{inc_2, inc_3\}$ $\{inc_2\}$
- IDR: $\{inc_1\}$ $\{inc_1\}$ $\{inc_1\}$ $\{inc_1, inc_2\}$ $\{inc_2, inc_3\}$ $\{inc_2, inc_3\}$

- Time
- $p_1$ enters room $x$ $p_1$ enters room $y$ $p_3$ enters room $y$ $p_3$ enters room $x$ $p_2$ leaves room $x$ $p_3$ leaves room $y$
1. Background and existing work
   - Context-aware applications
   - Constraint checking techniques
2. Problem
   - How to balance immediate scheduling and batch-based scheduling?
3. Our approach
   - Generic adaptive scheduling (GEAS)
4. Evaluation
   - Controlled experiments and real-world scenarios
5. Conclusion and future plan
Evaluation

1. Evaluation setup
   ❖ **Subject: SUTPC**\(^6\)
     - 21 consistency constraints
     - 1.6 million 24-hour taxi data
     - Over 4 million context changes

   ❖ **Scheduling strategies**
     - ImmedSched, BatchSched, GEAS

   ❖ **Checking techniques**
     - ECC\(^4\), PCC\(^6\), Con-C\(^5\), GAIN\(^8\)
Research question

1. RQ1: severe fact
   ❖ How severe is BatchSched’s inconsistency missing problem?

2. RQ2: efficiency (checking time)
   ❖ How efficient is GEAS in context inconsistency detection?

3. RQ3: effectiveness (inconsistency missing rate)
   ❖ How effective is GEAS in context inconsistency detection?
RQ1: severe fact

BatchSched can improve the efficiency of context inconsistency detection,

but it also caused a serious inconsistency missing problem

Fig. 5: Checking time comparison for BatchSched

Fig. 6: Missing rate comparison for BatchSched
RQ1: severe fact

BatchSched can improve the efficiency of context inconsistency detection,

but it also caused a serious inconsistency missing problem

Fig. 5: Checking time comparison for BatchSched

Fig. 6: Missing rate comparison for BatchSched
RQ2: efficiency

Fig. 8: Efficiency (checking time) comparison (in hour)
RQ2: efficiency

Fig. 8: Efficiency (checking time) comparison (in hour)

Fig. 7: Efficiency (checking time) comparison (normalized with ECC data as 100%).
**RQ2: efficiency**

GEAS can reduce the checking time greatly for all constraint checking techniques, e.g., 84.5% reduction for ECC, 84.4% for Con-C, 83.5% for GAIN and 29.9% for PCC.

![Efficiency comparison graph](Image)

**Fig. 8: Efficiency (checking time) comparison (in hour)**

![Efficiency comparison graph](Image)

**Fig. 7: Efficiency (checking time) comparison (normalized with ECC data as 100%).**
RQ3: effectiveness

GEAS is effective in context inconsistency detection and completely avoid missing context inconsistencies in the detection.

BatchSched can cause a 39.2-65.3% missing rate.

Fig. 9: Missing rate comparison
Real-world scenarios

1. Simulate real-world scenarios
Real-world scenarios

1. Simulate real-world scenarios
   - Processing context changes according to the actual timestamps

<table>
<thead>
<tr>
<th></th>
<th>ImmedSched</th>
<th>BatchSched</th>
<th>GEAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Missing rate</td>
<td>0-95.4%</td>
<td>39.2-67.7%</td>
<td>0%</td>
</tr>
</tbody>
</table>
Real-world scenarios

1. Simulate real-world scenarios
   - Processing context changes according to the actual timestamps

<table>
<thead>
<tr>
<th></th>
<th>ImmedSched</th>
<th>BatchSched</th>
<th>GEAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Missing rate</td>
<td>0-95.4%</td>
<td>39.2-67.7%</td>
<td>0%</td>
</tr>
</tbody>
</table>

2. Simulate extremely heavy-workload scenarios

<table>
<thead>
<tr>
<th></th>
<th>ImmedSched</th>
<th>BatchSched</th>
<th>GEAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Missing rate (except for ECC)</td>
<td>93.1-97.8% (ECC: 0.9% detected)</td>
<td>49.4%-92.2% (ECC: 6.3% detected)</td>
<td>0% (ECC: 11% detected)</td>
</tr>
<tr>
<td>Efficiency improvement</td>
<td>100%</td>
<td>100-689%</td>
<td>106-935%</td>
</tr>
</tbody>
</table>
Outline

1. Background and existing work
   ❖ Context-aware applications
   ❖ Constraint checking techniques

2. Problem
   ❖ How to balance immediate scheduling and batch-based scheduling?

3. Our approach
   ❖ Generic adaptive scheduling (GEAS)

4. Evaluation
   ❖ Controlled experiments and real-world scenarios

5. Conclusion and future plan
Conclusion & future plan
Conclusion & future plan

1. Conclusion

- **Generic adaptive scheduling (GEAS)**
  - Improve the **efficiency** of inconsistency detection by sampling, with 1.4~6.5x speed up to existing checking techniques
  - **Avoid missing** context inconsistencies, which has also been theoretically proved by us
  - **Generic** to existing constraint checking techniques, by focusing on orthogonal dimensions
Conclusion & future plan

1. Conclusion
   ❖ **Generic adaptive scheduling (GEAS)**
     • Improve the **efficiency** of inconsistency detection by sampling, with 1.4~6.5x speed up to existing checking techniques
     • **Avoid missing** context inconsistencies, which has also been theoretically proved by us
     • **Generic** to existing constraint checking techniques, by focusing on orthogonal dimensions

2. Future plan
   ❖ **Refine GEAS to make it more precise**
     • GEAS uses a conservative way to derive suspicious pairs so far
     • Refinement is expected to bring further efficiency improvement
Thank you

Comments are welcome!

http://moon.nju.edu.cn/
Email: cocowhy1013@gmail.com
Reference

Appendix (1): summary

1. Inconsistency detection includes two orthogonal dimensions

- How:
  - checking technique:
    - ECC
    - PCC
    - Con-C
    - GAIN

- When:
  - scheduling strategy:
    - ImmedSched
    - BatchSched
    - GEAS
Appendix (2): algorithm

1. Matching and scheduling
   - Matching such combination against suspicious pairs
     - The newly collected context change
     - Existing context changes in a batch
   - Scheduling constraint checking if matched successfully
     - Deal with all context changes in the current batch
Appendix (2): algorithm

1. Matching and scheduling
   
   - Matching such combination against suspicious pairs
     - The newly collected context change
     - Existing context changes in a batch
   
   - Scheduling constraint checking if matched successfully
     - Deal with all context changes in the current batch

```
Algorithm 1 Matching changes against suspicious pairs

Input: S (all suspicious pairs), chg (a context change), 
setOfChgs (context changes in the current batch)
Output: result (whether a suspicious pair is matched)

1: for any change c in setOfChgs do
   2: if c.category == “inc+” or “inc?” then
   3:     if S contains (c, chg) then
   4:         result := TRUE;
   5:         break;
   6:     end if
   7: end if
8: end for
9: result := FALSE;
10: return result
```

Diagram showing the timeline of changes with "inc−".
Appendix (3): GEAS’ overhead

GEAS’ overhead can be ignored.

Measuring GEAS’ overhead