Mining Temporal Invariants from Partially Ordered Logs

Ivan Beschastnikh
Yuriy Brun
Michael D. Ernst
Arvind Krishnamurthy
Thomas E. Anderson

University of Washington
Motivating question

I am a developer.

Why does my system behave in a certain manner?
Synoptic (our prior work)

A tool that mines FSM models from logs

Output

Input

propose

commit

abort

tx-commit

tx-abort
Why does my concurrent system behave in a certain manner?
Log analysis of concurrent systems

- Concurrency is widespread and is becoming commonplace (Hadoop, Ajax, Multicore)
- Many log analysis tools exist to help understand sequential, but not concurrent systems
  - Assume totally ordered logs
  - Cannot reason about concurrent executions
  - Insufficient for debugging concurrency issues
Log analysis of concurrent systems

• Concurrency is widespread and is becoming commonplace (Hadoop, Ajax, Multicore)

• Many log analysis tools exist to help understand

   Need to develop tools for concurrent systems logs

   • Cannot reason about concurrent executions

   • Insufficient for debugging concurrency issues
Our approach

• **Mine the partially ordered log** to extract temporal invariants between events

  • Capture the essence of what happened
  • Simple to understand

• Show invariants to the developer

  • May notice missing invariants
  • May find unexpected invariants

• Developer modifies and re-runs the system
Outline

• Motivation

• Why a total order is not enough

• Mining temporal invariants from concurrent executions

• Tool demo

• Two algorithms to mine temporal invariants

• Algorithms’ scalability evaluation
Limitations of total order

- A system with two threads: T1, T2
  - T1 generates event \( a \), T2 generates event \( b \)

- Logging pipeline:
- Generated log file:

```
T1
\[ \text{Logger} \rightarrow \text{Log file} \]
T2

```

```
1 | a
2 | b
```
Limitations of total order

• A system with two threads: T1, T2

• T1 generates event \(a\), T2 generates event \(b\)

• Logging pipeline:

\[
\begin{array}{c}
\text{T1} \\
\text{Logger} \\
\text{T2}
\end{array} \xrightarrow{} \begin{array}{c}
\text{Log file}
\end{array}
\]

• Generated log file:

\[
\begin{array}{c|c}
1 & a \\
2 & b
\end{array}
\]

Which of these three systems generated the log?
Limitations of total order

• A system with two threads: T1, T2
  • T1 generates event \( a \), T2 generates event \( b \)

• Logging pipeline:

• Generated log file:
  
A totally ordered log is insufficient.
Logging the partial order

- We know how to do this
- Lamport defined the happens-before relation in 1978
- Operationalized with vector clocks in 1988, 1989
Example system

- A server with tickets, two clients who buy tickets
- Each client checks availability of tickets and then buys a ticket
Partial order is complex

Partially ordered log:

[1,0,0] client 0: search for tickets to Portugal for 23/10/11
[0,1,0] client 1: search for tickets to Portugal for 23/10/11
[1,0,1] server: there is a ticket available for 505P
[1,1,2] server: there is a ticket available for 505P
[2,0,1] client 0: buy ticket
[2,1,3] server: sold
[1,2,2] client 1: buy ticket
[2,2,4] server: tickets sold out

Execution:

```
[1,0,0] client 0: search for tickets to Portugal for 23/10/11
[0,1,0] client 1: search for tickets to Portugal for 23/10/11
[1,0,1] server: there is a ticket available for 505P
[1,1,2] server: there is a ticket available for 505P
[2,0,1] client 0: buy ticket
[2,1,3] server: sold
[1,2,2] client 1: buy ticket
[2,2,4] server: tickets sold out
```
Partial order is complex

Partially ordered log:

- [1,0,0] client 0: search for tickets to Portugal for 23/10/11
- [0,1,0] client 1: search for tickets to Portugal for 23/10/11
- [1,0,1] server: there is a ticket available for 505P
- [1,1,2] server: there is a ticket available for 505P
- [2,0,1] client 0: buy ticket
- [2,1,3] server: sold
- [1,2,2] client 1: buy ticket
- [2,2,4] server: tickets sold out

- [1,0,0] client 0: search for tickets to Portugal for 23/10/11
- [1,0,1] server: there is a ticket available for 505P
- [2,0,1] client 0: buy ticket
- [2,0,2] server: sold
- [0,1,0] client 1: search for tickets to Portugal for 23/10/11
- [2,1,3] server: tickets sold out

- [0,1,0] client 1: search for tickets to Portugal for 23/10/11
- [1,0,0] client 0: search for tickets to Portugal for 23/10/11
- [0,1,1] server: there is a ticket available for 505P
- [1,1,2] server: there is a ticket available for 505P
- [0,2,1] client 1: buy ticket
- [1,2,3] server: sold
- [2,1,2] client 0: buy ticket
- [2,2,4] server: tickets sold out

- [1,0,0] client 0: search for tickets to Portugal for 23/10/11
- [1,0,1] server: there is a ticket available for 505P
- [0,1,0] client 1: search for tickets to Portugal for 23/10/11
- [1,1,2] server: there is a ticket available for 505P
- [1,2,2] client 1: buy ticket
- [1,2,3] server: sold
- [2,0,1] client 0: buy ticket
- [2,2,4] server: tickets sold out

- [1,0,0] client 0: search for tickets to Portugal for 23/10/11
- [1,0,1] server: there is a ticket available for 505P
- [0,1,0] client 1: search for tickets to Portugal for 23/10/11
- [1,1,2] server: there is a ticket available for 505P
- [1,1,2] server: there is a ticket available for 505P
- [2,0,1] client 0: buy ticket
- [2,1,3] server: sold
- [1,2,2] client 1: buy ticket
- [2,2,4] server: tickets sold out

Executions:
Need a way to summarize a partially ordered log
Temporal invariants

• **Mine** the partially ordered log to extract temporal invariants between events

• Temporal invariants
  • True for all logged executions
    • Capture the essence of what happened
  • Simple to understand
    • Each invariants involves at most two hosts
  • Summarize the partial order
## Five temporal log invariants

<table>
<thead>
<tr>
<th>Invariant</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 \rightarrow y_2$</td>
<td>liveness</td>
</tr>
<tr>
<td>always followed by</td>
<td>safety</td>
</tr>
<tr>
<td>$x_1 \leftarrow y_2$</td>
<td>safety</td>
</tr>
<tr>
<td>always precedes</td>
<td></td>
</tr>
<tr>
<td>$x_1 \rightarrow y_2$</td>
<td>safety</td>
</tr>
<tr>
<td>never followed by</td>
<td></td>
</tr>
<tr>
<td>$x_1 \parallel y_2$</td>
<td>safety</td>
</tr>
<tr>
<td>always concurrent with</td>
<td></td>
</tr>
<tr>
<td>$x_1 \perp y_2$</td>
<td>safety</td>
</tr>
<tr>
<td>never concurrent with</td>
<td></td>
</tr>
</tbody>
</table>

**Execution 1**

- T1: $x$
- T2: $y$

**Execution 2**

- T1: $a$
- T2: $b$

**Execution 3**

- T1: $a$
- T2: $y$
### Five temporal log invariants

<table>
<thead>
<tr>
<th>Invariant</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 \rightarrow y_2$ always followed by</td>
<td>liveness</td>
</tr>
<tr>
<td>$x_1 \leftarrow y_2$ always precedes</td>
<td>safety</td>
</tr>
<tr>
<td>$x_1 \rightarrow y_2$ never followed by</td>
<td>safety</td>
</tr>
<tr>
<td>$x_1 \parallel y_2$ always concurrent with</td>
<td>safety</td>
</tr>
<tr>
<td>$x_1 \perp y_2$ never concurrent with</td>
<td>safety</td>
</tr>
</tbody>
</table>

**Execution 1**

```
T1 ───> T2
x    y
```

**Execution 2**

```
T1 ───> T2
a    y
```

**Execution 3**

```
T1 ───> T2
a    b
```
## Five temporal log invariants

<table>
<thead>
<tr>
<th>Invariant</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 \rightarrow y_2$</td>
<td>liveness</td>
</tr>
<tr>
<td>always followed by</td>
<td></td>
</tr>
<tr>
<td>$x_1 \leftarrow y_2$</td>
<td>safety</td>
</tr>
<tr>
<td>always precedes</td>
<td></td>
</tr>
<tr>
<td>$x_1 \nLeftarrow y_2$</td>
<td>safety</td>
</tr>
<tr>
<td>never followed by</td>
<td></td>
</tr>
<tr>
<td>$x_1 \parallel y_2$</td>
<td>safety</td>
</tr>
<tr>
<td>always concurrent with</td>
<td></td>
</tr>
<tr>
<td>$x_1 \nLeftarrow y_2$</td>
<td>safety</td>
</tr>
<tr>
<td>never concurrent with</td>
<td></td>
</tr>
</tbody>
</table>

### Execution 1

![Execution 1 Diagram](image1.png)

### Execution 2

![Execution 2 Diagram](image2.png)
## Five temporal log invariants

<table>
<thead>
<tr>
<th>Invariant</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 \rightarrow y_2$</td>
<td>liveness</td>
</tr>
<tr>
<td>always followed by</td>
<td></td>
</tr>
<tr>
<td>$x_1 \leftarrow y_2$</td>
<td>safety</td>
</tr>
<tr>
<td>always precedes</td>
<td></td>
</tr>
<tr>
<td>$x_1 \nleftrightarrow y_2$</td>
<td>safety</td>
</tr>
<tr>
<td>never followed by</td>
<td></td>
</tr>
<tr>
<td>$x_1 \nparallel y_2$</td>
<td>safety</td>
</tr>
<tr>
<td>always concurrent with</td>
<td></td>
</tr>
<tr>
<td>$x_1 \not\nparallel y_2$</td>
<td>safety</td>
</tr>
<tr>
<td>never concurrent with</td>
<td></td>
</tr>
</tbody>
</table>

### Execution 1

- T1: x
- T2: y

### Execution 2

- T1: a
- T2: y
- T3: b
## Five temporal log invariants

<table>
<thead>
<tr>
<th>Invariant</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 \rightarrow y_2$</td>
<td>liveness</td>
</tr>
<tr>
<td>always followed by</td>
<td></td>
</tr>
<tr>
<td>$x_1 \leftarrow y_2$</td>
<td>safety</td>
</tr>
<tr>
<td>always precedes</td>
<td></td>
</tr>
<tr>
<td>$x_1 \rightarrow y_2$</td>
<td>safety</td>
</tr>
<tr>
<td>never followed by</td>
<td></td>
</tr>
<tr>
<td>$x_1 \parallel y_2$</td>
<td>safety</td>
</tr>
<tr>
<td>always concurrent with</td>
<td></td>
</tr>
<tr>
<td>$x_1 \Leftrightarrow y_2$</td>
<td>safety</td>
</tr>
<tr>
<td>never concurrent with</td>
<td></td>
</tr>
</tbody>
</table>

Explanations:
- **Liveness** (always followed by): Event $x_1$ always happens before event $y_2$.
- **Safety**: Event $x_1$ always precedes event $y_2$.
- **Safety**: Event $x_1$ never happens after event $y_2$.
- **Safety**: Event $x_1$ and event $y_2$ always happen at the same time.
- **Safety**: Event $x_1$ never happens at the same time as event $y_2$.

**Diagrams**:
- **Execution 1**: Event $x$ always follows event $y$.
- **Execution 2**: Event $x$ always precedes event $y$.
- **Execution 3**: Event $x$ never concurrent with event $y$.
## Mined invariants

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Invariants</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AlwaysFollowedBy</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INITIAL, search_client-0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>search_client-0, available_server</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INITIAL, search_client-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INITIAL, available_server</td>
<td></td>
<td></td>
</tr>
<tr>
<td>buy_client-0, sold-out_server</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sold_server, sold-out_server</td>
<td></td>
<td></td>
</tr>
<tr>
<td>buy_client-1, sold-out_server</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>NeverFollowedBy</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>search_client-0, search_client-0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>available_server, search_client-0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>buy_client-0, search_client-0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sold_server, search_client-0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>buy_client-1, search_client-0</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>AlwaysPrecedes</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>search_client-0, buy_client-0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>search_client-0, sold_server</td>
<td></td>
<td></td>
</tr>
<tr>
<td>search_client-0, sold-out_server</td>
<td></td>
<td></td>
</tr>
<tr>
<td>search_client-1, buy_client-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>search_client-1, sold-out_server</td>
<td></td>
<td></td>
</tr>
<tr>
<td>available_server, buy_client-0</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>AlwaysConcurrentWith</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>search_client-1, search_client-0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>buy_client-1, buy_client-0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>available_server, buy_client-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>available_server, sold_server</td>
<td></td>
<td></td>
</tr>
<tr>
<td>available_server, sold-out_server</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Mined invariants

- Always followed by: available\(_s\) ← buy\(_{c_0}\)
- Always precedes: available\(_s\) ← buy\(_{c_1}\)
- Never followed by: sold-out\(_s\) ⊥ sold\(_s\)
- Always concurrent with: sold\(_s\) ← sold-out\(_s\)
- Never concurrent with: buy\(_{c_0}\) → sold-out\(_s\)

Other invariants:
- sold-out\(_s\) ⊥ buy\(_{c_0}\)
- sold\(_s\) ← sold-out\(_s\)
- buy\(_{c_0}\) || buy\(_{c_1}\)
- search\(_{c_0}\) || search\(_{c_1}\)
- buy\(_{c_0}\) ← sold-out\(_s\)
# Mined invariants

<table>
<thead>
<tr>
<th>always followed by</th>
<th>always precedes</th>
<th>never followed by</th>
<th>always concurrent with</th>
<th>never concurrent with</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Server event</th>
<th>Client events</th>
</tr>
</thead>
<tbody>
<tr>
<td>available_s ← buy_{c_0}</td>
<td>sold-out_{s} ↯ sold_{s}</td>
</tr>
<tr>
<td>available_s ← buy_{c_1}</td>
<td>sold_{s} ← sold-out_{s}</td>
</tr>
<tr>
<td>sold-out_{s} ↯ buy_{c_0}</td>
<td>buy_{c_0} ↦ buy_{c_1}</td>
</tr>
<tr>
<td>sold-out_{s} ↯ buy_{c_1}</td>
<td>search_{c_0} ↦ search_{c_1}</td>
</tr>
<tr>
<td>buy_{c_0} ← sold-out_{s}</td>
<td>buy_{c_1} ← sold-out_{s}</td>
</tr>
<tr>
<td>buy_{c_0} → sold-out_{s}</td>
<td>buy_{c_1} → sold-out_{s}</td>
</tr>
</tbody>
</table>
Mined invariants

always followed by  always precedes  never followed by  always concurrent with  never concurrent with

\[
\begin{align*}
\text{available}_s &\leftarrow \text{buy}_{c_0} \\
\text{available}_s &\leftarrow \text{buy}_{c_1} \\
\text{sold-out}_s &\nleftrightarrow \text{buy}_{c_0} \\
\text{sold-out}_s &\nleftrightarrow \text{buy}_{c_1}
\end{align*}
\]

Temporal orderings between server and client events
Mined invariants

<table>
<thead>
<tr>
<th>always followed by</th>
<th>always precedes</th>
<th>never followed by</th>
<th>always concurrent with</th>
<th>never concurrent with</th>
</tr>
</thead>
<tbody>
<tr>
<td>available$<em>s$ ← buy$</em>{c0}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>available$<em>s$ ← buy$</em>{c1}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sold-out$<em>s$ ↗ buy$</em>{c0}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sold-out$<em>s$ ↗ buy$</em>{c1}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Server-side correctness invariants

<table>
<thead>
<tr>
<th>buy$_{c0}$ → sold-out$_s$</th>
<th>buy$_{c1}$ → sold-out$_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>buy$_{c0}$</td>
<td></td>
</tr>
<tr>
<td>search$_{c0}$</td>
<td></td>
</tr>
<tr>
<td>buy$_{c0}$ ← sold-out$_s$</td>
<td>buy$_{c0}$ ← sold-out$_s$</td>
</tr>
</tbody>
</table>

Temporal orderings between server and client events
Mined invariants

always followed by | always precedes | never followed by | always concurrent with | never concurrent with

| available_s ← buy_{c_0} | | | | |
| available_s ← buy_{c_1} | | | | |
| sold-out_s \not\leftrightarrow buy_{c_0} | | | | |
| sold-out_s \not\leftrightarrow buy_{c_1} | | | | |

Server-side correctness invariants

sold-out_s \not\leftrightarrow sold_s
sold_s ← sold-out_{s_{\text{\_s}}} | run_{c_0} || run_{c_1} |
search_{c_0} || search_{c_1} |
buy_{c_0} \rightarrow sold-out_{s_{\text{\_s}}} | run_{c_0} || run_{c_1} |
buy_{c_1} \rightarrow sold-out_{s_{\text{\_s}}} | run_{c_0} || run_{c_1} |
buy_{c_0} ← sold-out_{s_{\text{\_s}}} | run_{c_0} || run_{c_1} |

Temporal orderings between server and client events

Concurrency between clients
Mined invariants

<table>
<thead>
<tr>
<th>always followed by</th>
<th>always precedes</th>
<th>never followed by</th>
<th>always concurrent with</th>
<th>never concurrent with</th>
</tr>
</thead>
<tbody>
<tr>
<td>available_s ← buy_{c_0}</td>
<td></td>
<td>sold-out_s ↦ sold_s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>available_s ← buy_{c_1}</td>
<td></td>
<td>sold_s ← sold-out_s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sold-out_s ↦ buy_{c_0}</td>
<td></td>
<td>buy_{c_0}</td>
<td></td>
<td>buy_{c_1}</td>
</tr>
<tr>
<td>sold-out_s ↦ buy_{c_1}</td>
<td></td>
<td>search_{c_0}</td>
<td></td>
<td>search_{c_1}</td>
</tr>
<tr>
<td>buy_{c_0} → sold-out_s</td>
<td></td>
<td>buy_{c_1} → sold-out_s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>buy_{c_0} ← sold-out_s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Temporal orderings between server and client events

Concurrency between clients

Over-fit invariants
Outline

• Motivation

• Why a total order is not enough

• Mining temporal invariants from concurrent executions

• Tool demo

• Two algorithms to mine temporal invariants

• Algorithms’ scalability evaluation
Algorithms to mine invariants

1. An algorithm based on the transitive closure
2. A co-occurrence counting algorithm (v1)
3. A modified co-occurrence counting algorithm (v2) that omits “never concurrent with”

More details in the paper
Transitive closure mining

• Compute the transitive closure of all execution DAGs
• Use the transitive closure to compute invariants

\[ x_1 \rightarrow y_2 \]
always followed by

Log

Transitive Closure

Invariants
Co-occurrence counting mining

- Count the number of times events co-occur
- Use counts to compute invariants

![Log Diagram]

<table>
<thead>
<tr>
<th>event</th>
<th>total count</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>3</td>
</tr>
<tr>
<td>y2</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>event pair</th>
<th># co-occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1,y2</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

\[ x_1 \rightarrow y_2 \]
always followed by
Evaluation methodology

- A discrete time simulator of a distributed system with $H$ hosts that use vector clocks to maintain a partial order
- Each host generates a total of $E$ events
- Each event is one of $T$ types
- Hosts communicate with probability 0.3
- Invariants are mined from the resulting log

Vary each variable to evaluate algorithm scalability
Scalability results

- Time (s) vs. Number of executions
- Time (s) vs. Nodes in the system
- Time (s) vs. Number of event types
- Time (s) vs. Length of an execution trace

Graphs show the comparison of Transitive Closure and Co-occurrence Counting v1 and v2 for different metrics.
Limitations and future work

• Logging the partial order explicitly has a performance penalty: extra network traffic/computation/state

  • Has been previously studied
    
    Charron-Bost IPL 1991
    Khotimsky and Zhuklinets ICATM 1999

• Invariants are a summary and do not provide a complete view

  Dwyer et al. ICSE 1999

• Visualization of distributed traces

  Edwards et al. IPDPS 1994
Conclusion

• Studying logs of concurrent systems is becoming increasingly important

• Temporal invariants can help explain a complex concurrent system log

• Presented algorithms to mine five types of temporal invariants

Try it!
http://synoptic.googlecode.com