Tracing and Monitoring Distributed Multi-Core Systems Project
- Progress Meeting -

User Space Trace Abstraction Techniques
Heidar Pirzadeh and Wahab Hamou-Lhadj
Software Behaviour Analysis Lab
{s_pirzad, abdelw}@ece.concordia.ca

Montreal, QC
Dec. 09, 2011
Progress

• Trace abstraction:
  – We continued to develop trace abstraction techniques for user space traces
  – Explored the use of state information in trace abstraction and exploration
  – Developed techniques for automatically extracting important content from a trace

• Anomaly detection:
  – Investigation of different tracing mechanisms
  – Reduction of learning time in building models
  – Reduction of false positives
  – Development of a taxonomy of attacks on the Linux kernel
Our Approach for Trace Abstraction

• Based on the extraction of execution phases from large traces

• What is an execution phase?
  – A segment of program’s execution that performs a specific task

• Trace Segmentation: Automatically divide a trace into phases
  – Allow SW engineering to browse traces as a flow of execution phases rather than mere sequence of events
Example

• A trace generated from a compiler will contain the various compiler’s phases including parsing, preprocessing, lexical analysis, semantic analysis, etc.

• In most visualization tools, it will look like:

• But how can we tell what happens where?
Visually…
Visually…

<table>
<thead>
<tr>
<th>Init</th>
<th>Parsing</th>
<th>Preprocessing</th>
<th>Lexical Analysis</th>
<th>Semantic Analysis</th>
</tr>
</thead>
</table>

[Table content not visible in the image]
A different view…
Nested phases can be added
Research Questions?

• How can we automatically extract execution phases from a trace?
• What additional information states can reveal about execution phases?
• How can we extract the main components that implement a specific phase?
• Can we use execution phases to further reduce the size of traces?
Approach: Trace Abstraction Framework
Our Approach: Trace Abstraction Framework
Trace Segmentation Approach

• The scientific foundation comes from the study of the human perception system
  – The ability for humans to group similar items to form objects and shapes
  – Explained using the Gestalt laws of similarity and continuity
Measuring Similarity
Measuring Continuity in Traces with Nesting Levels
Measuring Continuity in Traces with Nesting Levels
Case Study

**Program:** WEKA 3.6.6

**Scenario:** building a decision tree learning algorithm for classifying data instances.

**Trace:** Multi-threads 1,571,214 events
Phase flow diagram of a Weka trace
Adding phase views to a tool
State Information

• **What is a state?**
  – The state of the system is the state value of every attribute in the system
  – State has a duration
  – State value, which can really be anything

• **Attributes in the kernel-trace state system:**
  – CPUs
  – CPUs/0
  – CPUs/0/current_thread
  – Etc.

State Change

Consists of three things:

- timestamp
- attribute
- state value

The state of 'attribute' changed to 'state value' at time 'timestamp'

Existing Info

LTTNG Kernel Space Trace:
• Timestamp
• Event (page fault)
• Process ID
• CPU ID
• File Descriptor
Phase Flow

Threads

Phases

P1

P2

P3

P4

P5

P6

P7

P8

P9

P10

P11

P12

P13

P14

P15
Phases Mapped to Kernel Space Trace
Phases Mapped to Kernel Space Trace

Threads

- t1: P1
- t2: P2, P3, P4, P5, P6
- t3: P7, P8, P9
- t4: P10, P11, P12, P13, P14, P15

Timestamps:
- $t_1$ and $t_2$ for t1
- $t$ and $t'$ for t2
## Phases Enriched with State Info

### Threads

<table>
<thead>
<tr>
<th>PID</th>
<th>CPU</th>
<th>FD</th>
<th>Page Fault Ratio</th>
<th>CPU usage</th>
<th>Mem. Usage</th>
</tr>
</thead>
</table>

### Time Stamps
- **t1**:
  - P1

- **t2**:
  - P2
  - P3
  - P4
  - P5
  - P6

- **t3**:
  - P7
  - P8
  - P9

- **t4**:
  - P10
  - P11
  - P12
  - P13
  - P14
  - P15

### Time Map
- **timestamp : t**: T
- **timestamp : t’**: T’
Phases Enriched: Statistics (1)

Threads

|CPU|: 2  
|PID|: 15  
|FD|: 14  
|PageFault|: 453  
|Ratio|: 60.06 %
Phases Enriched: Statistics (2)

Threads

- t1: P1
- t2: P2, P3, P4, P5, P6
- t3: P7, P8, P9
- t4: P10, P11, P12, P13, P14, P15

<table>
<thead>
<tr>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PID</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FD</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PageFault</th>
</tr>
</thead>
<tbody>
<tr>
<td>526</td>
</tr>
</tbody>
</table>

Ratio: 15.03%

<table>
<thead>
<tr>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PID</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FD</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PageFault</th>
</tr>
</thead>
<tbody>
<tr>
<td>453</td>
</tr>
</tbody>
</table>

Ratio: 60.06%
Comparison: Kernel Space vs User Space
Enriched Phase View
Approach: Trace Abstraction Framework
Content Prioritization

1. Extract representative elements of each phase

- Can give a hint about what is happening in a phase
- Uncover the most relevant elements that implement the traced scenario
Content Prioritization

2- Finding similar phases

- Can give a hint about what is happening in a phase
- Uncover the most relevant elements that implement the traced scenario
Content Prioritization

2- Finding similar phases

- Can give a hint about what is happening in a phase
- Uncover the most relevant elements that implement the traced scenario
- Optimized flow of phases
Extracting Relevant Components

- Idea: Elements that are repeated in a phase but are not much shared between phases indicate their relevance to the phase

- This is similar to the concept of term frequency inverse document frequency in the text mining

Document 1: Shipment of gold damaged in a fire
Document 2: Delivery of silver arrived in a silver truck
Document 3: Shipment of gold arrived in a truck
## Extracting Representative Elements

<table>
<thead>
<tr>
<th>Trace T</th>
<th>(L_{ik})</th>
<th>(IG_i)</th>
<th>(w_{ik})</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>L(a)=1</td>
<td>G(a)=0.17</td>
<td>v(a)=0.45</td>
</tr>
<tr>
<td>d</td>
<td>L(d)=1.3</td>
<td>G(d)=0.17</td>
<td>w(d)=0.58</td>
</tr>
<tr>
<td>c</td>
<td>L(c)=1.4</td>
<td>G(c)=0.17</td>
<td>v(c)=0.66</td>
</tr>
<tr>
<td>i</td>
<td>L(i)=1</td>
<td>G(i)=0</td>
<td>w(i)=0</td>
</tr>
<tr>
<td>e</td>
<td>L(e)=1</td>
<td>G(e)=0</td>
<td>w(e)=0</td>
</tr>
<tr>
<td>h</td>
<td>L(h)=1</td>
<td>G(h)=0.47</td>
<td>v(h)=0.50</td>
</tr>
<tr>
<td>i</td>
<td>L(i)=1</td>
<td>G(i)=0</td>
<td>w(i)=0</td>
</tr>
<tr>
<td>m</td>
<td>L(m)=1.3</td>
<td>G(m)=0.47</td>
<td>v(m)=0.65</td>
</tr>
<tr>
<td>e</td>
<td>L(e)=1</td>
<td>G(e)=0</td>
<td>w(e)=0</td>
</tr>
<tr>
<td>o</td>
<td>L(o)=1.3</td>
<td>G(o)=0.17</td>
<td>v(o)=0.24</td>
</tr>
<tr>
<td>a</td>
<td>L(a)=1</td>
<td>G(a)=0.17</td>
<td>v(a)=0.41</td>
</tr>
<tr>
<td>o</td>
<td>L(o)=1</td>
<td>G(d)=0.17</td>
<td>w(d)=0.53</td>
</tr>
<tr>
<td>d</td>
<td>L(d)=1.3</td>
<td>G(c)=0.17</td>
<td>v(c)=0.60</td>
</tr>
<tr>
<td>c</td>
<td>L(c)=1.4</td>
<td>G(i)=0</td>
<td>w(i)=0</td>
</tr>
<tr>
<td>i</td>
<td>L(i)=1</td>
<td>G(e)=0</td>
<td>w(e)=0</td>
</tr>
<tr>
<td>e</td>
<td>L(e)=1</td>
<td>G(o)=0.17</td>
<td>w(o)=0.41</td>
</tr>
</tbody>
</table>

\[
w_{i,k} = \frac{L_{i,k}}{\sqrt{\sum_{j=1}^{N_k} [\log (ef_{j,k}) + 1] \cdot \log \left( \frac{N}{n_i} \right)}} \cdot IG_i
\]

\[
w_{"d"}, \text{Phase 1} = \frac{1.3 \cdot 0.17}{\sqrt{(0.17)^2 + (0.22)^2 + (0.26)^2}} = 0.45
\]

Information about element “c”

Information about element “m”

Information about element “c”
Relevant Events Snapshots
Case Study: Relevant Events

 Threads

|CPU|: 2  
|PID|: 15  
|FD|: 14  
|PageFault|: 453  
|Ratio|: 60.06 %  

|CPU|: 2  
|PID|: 17  
|FD|: 16  
|PageFault|: 526  
|Ratio: 15.03%  

CPU usage: 40%

- weka.core.Instance.value
- weka.gui.visualize.Messages.getString
- weka.core.Attribute.isString
- weka.gui.explorer.Messages.getInstance
- weka.gui.explorer.Processes.getInstance

36
Conclusions

- We showed trace abstraction techniques based on execution phases.
- We added state information to extracted phases.
- We presented techniques for identifying the most relevant components of each phase.
Thank you!