Dynamic Detection of Event Handlers

Steven P. Reiss
Brown University
**Server Application**

- Today’s complex systems
- **Interactions**
  - Messages
  - Replies
  - UI (w/ updates)
- **Response**
  - DB reply
  - Computation
- **Multiple threads**
What Is Interesting

• **Information per interaction**
  • Performance for particular events
    • Inputs, transactions, user actions, …
  • Outputs associated with particular inputs
  • Events that result from other events
    • Protocol model
  • Dependencies between interactions

• **Analysis per interaction**

• How threads are used
What Is Required

• Understanding event processing
  • How events are processed
  • When events are processed
  • Where events are processed
  • What events are processed
  • What happens when processing events

• Associating threads with events

• FIRST STEP:
  • IDENTIFY THE EVENT HANDLERS
What Is An Event Handler

• “An asynchronous callback subroutine that handles inputs received in a program” (Wikipedia)

• Code of the form

```
LOOP
    E = Get next event
    Process event E
END
```
Complications

• **Code to get event can have many forms:**
  • Get next event in a routine
  • Get next event from a queue
  • Get next event by waiting on interrupt/notify
  • Process the read in line (socket read)
    • Check for complete message, loop if not
  • Callback from user interface
  • Callback from asynchronous I/O
  • Observer pattern (publish-subscribe)
Complications

• **Code for event can have many forms**
  - Call one routine
  - Call multiple routines
  - Switch to detect event type
    - Then call appropriate routine for event type
  - Parse/decode the message, then call handler
  - Debugging/logging statements
  - ...
DYPER/DYMON

- Dynamic analysis of performance issues
  - Fixed overhead analysis
  - Works with multiple dimensions
    - CPU, Memory, I/O, Heap, Threads, Sockets, ...
    - Each represented by a proflet
- Proflets have two components
  - One collects data
    - Based on stack samples
    - Based on scheduled detailed analysis
  - One analyzes and reports the result
Reaction Proflet

- **Find event handlers**
  - Accumulate performance data per event
- **Components**
  - Data collection to find event handlers
    - Based on stack traces
  - Data analysis to identify the handlers
  - Performance analysis based on handlers
    - Counts of time spent in each
    - Detailed analysis by instrumenting the handlers
Data Collection

• **Detect callbacks**
  • User routines called from system code
    • Usually (when does it not work?)

• **Consolidating call information**
  • Build a trie of calls
  • Accumulated from all threads
  • Leaf is stack base
    • Children are routines called
    • Only do user routines
  • Keep state counts for each node
    • RUN, IO, WAIT
Data Collection Example

7/21/2008
Data Analysis

- **Handle callbacks**
- **Look for Patterns in the trie**

- **Ensure significance based on counts**
  - Both relative and absolute
  - Cutoffs determined experimentally
Results Example

<REACTI ON TOTTME='493.81' TOTSAMP='39489'>
<CALLBACK METHOD='edu.brown.cs.cs032.crawler.crawl.CrawlSwingParser$Callback@handleEndTag' />
<CALLBACK METHOD='edu.brown.cs.cs032.crawler.crawl.CrawlSwingParser$Callback@handleStartTag' />
<CALLBACK METHOD='edu.brown.cs.cs032.crawler.crawl.CrawlSwingParser$Callback@handleError' />
<CALLBACK METHOD='edu.brown.cs.cs032.crawler.crawl.CrawlSwingParser$Callback@handleText' />
<EVENT METHOD='edu.brown.cs.cs032.crawler.url.UrlHandler.saveHtml' TYPE='NODE_I_O' />
<EVENT METHOD='edu.brown.cs.cs032.crawler.url.UrlHandler.addRedirectUrl' TYPE='NODE_I_O' />
<EVENT METHOD='edu.brown.cs.cs032.crawler.url.UrlHandler.addProcessing' TYPE='NODE_I_O' />
<EVENT METHOD='edu.brown.cs.cs032.crawler.url.UrlHandler.saveUrls' TYPE='NODE_WAIT' />
<EVENT METHOD='edu.brown.cs.cs032.crawler.url.UrlHandler.openConnection' TYPE='NODE_I_O' />
<EVENT METHOD='edu.brown.cs.cs032.crawler.url.UrlHandler.setError' TYPE='NODE_I_O' />
<EVENT METHOD='edu.brown.cs.cs032.crawler.crawl.CrawlThread@readContents' TYPE='NODE_I_O' />
<EVENT METHOD='edu.brown.cs.cs032.crawler.url.UrlHandler.setRedirectHtml' TYPE='NODE_I_O' />
<EVENT METHOD='edu.brown.cs.cs032.crawler.crawl.CrawlParser@parse' TYPE='NODE_I_O' />
<EVENT METHOD='edu.brown.cs.cs032.crawler.url.UrlHandler.saveLinks' TYPE='NODE_I_O' />
</REACTI ON>
Results

• Works on a wide variety of applications
  • Peer-to-peer server
  • Particle simulator
  • Web crawler
  • Code search engine
  • Simple test cases

• Some extra items detected
  • Access$100, uses of reflection
  • Others that are effectively handlers, but not thought of as such
    • currentTimeMillis()
Applications: DYMON

- Reaction statistics in DYMON

![DYPER Summary](image)

### Event Handling

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>% Time</th>
<th>Time</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>edu.brown.cs.cs032.crawler.crawl.CrawlSwingParser</td>
<td>handleStartTag</td>
<td>145.7</td>
<td>65,631.00</td>
<td>6,319,903</td>
</tr>
<tr>
<td>edu.brown.cs.cs032.crawler.crawl.CrawlSwingParser</td>
<td>handleText</td>
<td>1.5</td>
<td>673.00</td>
<td>4,266,019</td>
</tr>
<tr>
<td>edu.brown.cs.cs032.crawler.crawl.CrawlThread</td>
<td>readContents</td>
<td>21.6</td>
<td>9,718.00</td>
<td>37,558</td>
</tr>
<tr>
<td>edu.brown.cs.cs032.crawler.url.UrlHandle</td>
<td>endProcessing</td>
<td>3.3</td>
<td>1,465.00</td>
<td>41,018</td>
</tr>
<tr>
<td>edu.brown.cs.cs032.crawler.url.UrlHandle</td>
<td>saveHtml</td>
<td>68.8</td>
<td>31,019.00</td>
<td>37,558</td>
</tr>
<tr>
<td>edu.brown.cs.cs032.crawler.crawl.CrawlMain</td>
<td>loadUrls</td>
<td>22.1</td>
<td>9,975.00</td>
<td>0</td>
</tr>
<tr>
<td>edu.brown.cs.cs032.crawler.url.UrlHandle</td>
<td>saveLinks</td>
<td>47.3</td>
<td>21,311.00</td>
<td>37,588</td>
</tr>
<tr>
<td>edu.brown.cs.cs032.crawler.crawl.CrawlParser</td>
<td>parse</td>
<td>255.8</td>
<td>115,245.00</td>
<td>37,588</td>
</tr>
<tr>
<td>edu.brown.cs.cs032.crawler.url.UrlHandle</td>
<td>setError</td>
<td>4.0</td>
<td>1,808.00</td>
<td>2,714</td>
</tr>
</tbody>
</table>
Other Applications

- Creating a model of thread behavior
  - Based on what events a thread deals with
  - Based on what each event does
    - Thread states: CPU, WAIT, IO, BLOCK, ...
  - Can be used to predict performance
    - Based on # threads, # processors, ...

- Building a model of message processing
  - What messages cause other messages
  - Determining the message protocols
Conclusion

• Simple dynamic analysis
  • Can identify event handlers
  • Can serve as a basis for event-based analysis
  • Very low overhead

• The TRIE-counting techniques generalize
  • Building model of thread processing
  • Common understanding of behavior
    • Statistical basis
    • That combines multiple threads or processes

• IT WORKS
Acknowledgements

• NSF support: CCR0613162
• Code is available
  • ftp://ftp.cs.brown.edu/u/spr/wadi.tar.gz
  • Part of the WADI project
Questions / Comments