Instrumenting Python with Kieker

Serafim Simonov, Thomas Duellmann, Reiner Jung, Sven Gundlach
Kieker Instrumentation

Manually

- Type the API at every place in the code by yourself.

Automatic

- Code weaving/ Aspect Oriented Programming (AOP)
- AspectJ or AspectC
- Compiler weaving
AOP in Python

Python Features
- Decorators
- Meta classes
- Everything is an object

Frameworks
- aspectLib
  - Testing
- Spring Python
  - Last commit in 2014
Import Process in Python

1. Import statement
2. Check if module is in sys.modules
   - Yes: Return
   - No: Construct module

Diagram:
- Import statement
- Module in sys.modules?
  - Yes: Return
  - No: Construct module
Find and Load Modules in Python

- **Module spec finders**
- **Module name, path, loader**
- **load the module**
Post Import Weaving

1. Load the module normally
   • The source code is executed
2. Apply the decorator to functions
Issues

Problems:
- Information about original objects is lost
- Side effects
- Overwriting by the user code is possible
Pre Import Weaving

1. Find the module
2. Apply changes
   1. Parse the source code in to AST
   2. Add the decorators „manually“
   3. Compile and execute the modified version of the module
3. Execute
Old Problems Fixed

- No information loss
- No overwriting at actual runtime
- Less side effects
- C Error exception if we instrument “google.protobuf.descriptor” in Tensorflow
  - Some side effects persist
  - Possibly hidden interaction
Evaluation

• Moobench
  • The same principle as in Moobench for Java

• Configurations:
  • Inactive Instrumentation
  • TCP Writer
  • File Writer
  • Null Writer
  • No Instrumentation
## Evaluation Results

<table>
<thead>
<tr>
<th>setup</th>
<th>Python</th>
<th></th>
<th>Java</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean</td>
<td>1.q</td>
<td>2.q</td>
<td>3.q</td>
</tr>
<tr>
<td>w/o</td>
<td>6.908</td>
<td>6.860</td>
<td>6.892</td>
<td>6.932</td>
</tr>
<tr>
<td>probe inactive - post</td>
<td>15.131</td>
<td>15.096</td>
<td>15.139</td>
<td>15.188</td>
</tr>
<tr>
<td>probe inactive - pre</td>
<td>15.137</td>
<td>15.042</td>
<td>15.079</td>
<td>15.121</td>
</tr>
<tr>
<td>null writer - pre</td>
<td>118.255</td>
<td>117.655</td>
<td>117.881</td>
<td>118.160</td>
</tr>
<tr>
<td>text writer - post</td>
<td>2681.863</td>
<td>2104.617</td>
<td>2292.797</td>
<td>3236.825</td>
</tr>
<tr>
<td>text writer - pre</td>
<td>2772.709</td>
<td>2782.738</td>
<td>2793.667</td>
<td>2808.349</td>
</tr>
<tr>
<td>tcp writer - pre</td>
<td>960.505</td>
<td>870.799</td>
<td>876.135</td>
<td>1149.236</td>
</tr>
</tbody>
</table>
Summary

• **Current State**
  - The instrumentation techniques can be applied in other contexts
  - Two automatic instrumentation modes
  - Assumption: entry point of the Python program is known

• **Limitations**
  - Relies on „monkey patching“
  - The instrumentation can be overwritten and turned off
    - Client code overwrites `__import__()`
    - Client code changes `sys.meta_path`
  - No instrumentation of modules written in C
Future Work

• Improve efficiency
  • Writing of the records blocks the execution of the whole program

• Monitoring of `sys.meta_path`

• Instrumentation before execution
  • Requires extensive knowledge of Python grammar and return behavior

• Instrument modules written in C