Overview

- Many bugs result from mismatches between actual and expected system behavior
  - Structural – inappropriate order or placement of processing and/or communication
  - Performance – resource consumption or delays higher or lower than expected
- Pip reports behavior that violates expectations – potential bugs
- To use Pip, the programmer writes:
  - Annotations – additions to system source code to trace relevant activity
  - Expectations – description of expected system behavior in high-level language

Bacon: a simple example system

- Oracle of Bacon – find shortest paths between actors
- One server, many clients
- Significant CPU processing on server
- Up to 2 seconds per query

Table: Bacon: a simple example system

<table>
<thead>
<tr>
<th>User</th>
<th>Web Server</th>
<th>Query</th>
<th>Bacon Server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGI bin</td>
<td>Compute Shortest Path</td>
<td>Result</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Annotations

- Small source-code changes to important modules
- Highlight key communication and processing

Client annotations

- ANNOTATE_INIT(): pathid = get_time_in_microseconds();
- ANNOTATE_GET_PATH_ID(pathid);
- ANNOTATE_START_TASK("link people");
- msgid = get_time_in_microseconds();
- query = parse_user_query();
- ANNOTATE_RECEIVE(msgid);
- index_A = lookup_name(query.name_A);
- ANNOTATE_END_TASK("link people");

Server annotations

- ANNOTATE_END_PATH_ID(pathid);
- ANNOTATE_GET_PATH_ID(pathid);
- ANNOTATE_START_TASK("build BFS tree");
- ANNOTATE_END_TASK("build BFS tree");
- ANNOTATE_NOTICE("cache hit");
- ANNOTATE_NOTICE("cache miss");
- ANNOTATE_END_TASK("build BFS tree");
- ANNOTATE_NOTICE("shortest path");
- ANNOTATE_NOTICE("build BFS tree");
- ANNOTATE_NOTICE("shortest path");

Experience

- Bacon annotations: 84 lines for 1919 lines of source
- Other systems:
  - Bullet: 77 lines of annotations, 6425 lines of source
  - RanSub: 17 lines of annotations, 1711 lines of source
  - SplitStream: 108 lines of annotations, 3118 lines of source
  - Typically ~10 lines per 1000

Using Pip

1. Annotate application source code
2. Write expectations for the application
3. Run application to collect traces of actual behavior
4. Use Pip to import recognized events into SQL database
5. Run Pip’s checker to check application behavior against expectations
6. Explore some or all application behavior

Checking and exploring can be done off-line or in real time

Expectations

- Expectations file describes how to recognize and classify paths
  - Separate from application source code
  - Paths not recognized by any expectation may indicate bugs
  - Aggregate expectations apply to sets of paths

Sample expectations

<table>
<thead>
<tr>
<th>Validator</th>
<th>ShortestPath</th>
</tr>
</thead>
<tbody>
<tr>
<td>thread (Client, 1) task(&quot;link people&quot;) send(Serve) limit(CPU_TIME, 400ms);</td>
<td></td>
</tr>
<tr>
<td>recv(Serve)</td>
<td></td>
</tr>
<tr>
<td>thread (Server, 1) task(&quot;build path&quot;) recv(Client) repeat between 0 and 2 task(&quot;find person&quot;) limit(CPU_TIME, 400ms);</td>
<td></td>
</tr>
<tr>
<td>branch notice(&quot;cache miss&quot;) print(&quot;build BFS tree&quot;) limit(CPU_TIME, 1.0s);</td>
<td></td>
</tr>
<tr>
<td>branch notice(&quot;cache hit&quot;) branch () person not found notice(&quot;path length=16&quot;, i);</td>
<td></td>
</tr>
<tr>
<td>send(Client)</td>
<td></td>
</tr>
<tr>
<td>assert(average(CPU_TIME, ShortestPath) &lt; 0.6s); assert( instances(CacheMiss) / instances(CacheHit) &gt;= 0.9);</td>
<td></td>
</tr>
</tbody>
</table>

Resource metrics

- Real time, CPU time, page faults, context switches
- Message count, size, and latency
- Tree height, thread count

Exploring behavior

- Pick a task, host, path, or set of paths to examine
- Constrain view to a limited time range
- See one path or aggregate performance information

Sources of unexpected behavior

- Actual bugs: errors in application source code
- Incorrect annotations: too many, too few, wrong IDs
- Incorrect expectations

Oracle of Bacon

- Found two instances of unexpected behavior
- Operations with high delay but low CPU time
- Blocked on I/O due to single-threaded server design
- Unusually low cache-hit rate for one trace

Other systems

- RanSub
  - Erroneous start-up behavior
  - Some variables uninitialized
  - Causes extra messages and notices – unexpected structure
  - Excessive CPU time in recent reimplementation

- Bullet
  - Truncated paths at the end of the trace
  - Some messages lost – unexpected structure
  - Not a bug worth fixing

- SplitStream
  - Heartbeat paths with hundreds of events
  - Path IDs reused – annotation error