PerfGuard: Binary-Centric Application Performance Monitoring in Production Environments

Chung Hwan Kim, Junghwan Rhee*, Kyu Hyung Lee†, Xiangyu Zhang, Dongyan Xu

*NEC Laboratories America
Relentless passion for innovation

+The University of Georgia
Performance Problems

Amazon Web Services struck by performance issues

How performance issues brought down Chase's online banking site

GitHub recovers from major outage; cause unknown

Performance Issues for Yahoo

Yahoo's web sites experienced brief performance problems earlier today, with significant problems than others. Yahoo is one of the world’s busiest web sites users of the Netcraft toolbar), and thus the outages have been noted by the In yahoo.com home page was inaccessible for several hours from our London mo than usual from several locations in the U.S. Yahoo search appears to have ex
Performance Diagnosis During Development

- Complex dependencies and layers [PLDI '12]
- Various usage scenarios
- Limited testing environments

Performance diagnosis during production?
Performance Diagnosis in Production

- Software users do not have:
  - Source code
  - Development knowledge
- But, desire to analyze performance problems [SIGMETRICS ’14]
  - Many users are service providers
  - 3rd-party components

- Profilers and tracers:
  - perf
  - Callgrind
  - Ftrace
  - OProfile
  - Gprof
  - gperftools
  - LTTng

- CPU usage sampling
  - Constant overhead
  - Blind to program semantics
  - Sampling frequency
Performance Diagnosis in Production

- PerfTrack: Microsoft products only
- Application Performance Management (APM)
  - Limited # of pre-instrumented programs
  - Manual instrumentation with APIs
- Required: Source code and development knowledge

[Images of dynatrace, AppDynamics, and New Relic logos]
Automated Perf. Diagnosis in Production?

• Performance diagnosis without source code and development knowledge?
• At what granularity should we measure performance?
• When and where should we check performance?
• How can we determine if program is too slow?
Key Ideas

• Extract “hints” from program binaries through dynamic analysis

• Use the hints to identify individual operations (units)

• Generate and inject performance checks
PerfGuard: Binary-Centric Performance Check Creation and Injection

- Unit and Performance Guard Identification
- Instrumenting Program with Performance Guards
- Unit Performance Inspection
- Unit Performance Monitoring
- Feedback
- Profile
- Pre-distribution
- Production-run
- Deploy
- Assert (Latency <= Threshold)
Unit Identification

- **Unit** := One iteration of event processing loop [NDSS '13]

1. Most large-scale apps are event-driven
2. Small number of event processing loops

- **Type I**: UI programs
- **Type II**: Server programs
Unit Classification Based on Control Flow

• Units with different call trees have distinct performance

• Threshold estimation: based on time samples of unit groups
  • Average of 11% deviation in top 10 costly unit groups
Unit Clustering

- Hierarchical clustering
- Unit distance:
  \[
  \frac{\max(|p_i|,|p_j|) - LCS(p_i,p_j)}{\max(|p_i|,|p_j|)}
  \]
- Unit type:
  Set of clustered units
Performance Guard Generation

- 3 shared library functions
- Input: unit performance profile

OnLoopEntry (...) {
  u = NewUnit (...)
}

OnUnitStart (...) {
  t = NewTimer ()
}

Thread (...) {
  ...
  while (...) {
    Wait (e)
    Process (job)
    } // end while
  ...
  } // end Thread

OnUnitContext (...) {
  x = GetUnitType (...)
  Assert (t.Elapsed <= x.Threshold)
}

Perf. Profile
How to Recognize Unit Types at Run-Time

- Unit type election: Mark # of total occurrences

**Example: Unit Type X**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>D</th>
<th>G</th>
<th>E</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X:</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Y:</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>W:</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Z:</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Unit Type Candidates:
- (X, Y, W, Z)
- (X, Y, W, Z)
- (X, W, Z)
- (X)
- (X)
- (X)

**Diagram**

- Node A: (X, Y, W, Z)
- Node B: (X, Y, W, Z)
- Node C: (X, Y, W, Z)
- Node D: (X, W, Z)
- Node E: (X, Y, W)
- Node F: (Z)
- Node G: (X)
- Node H: (Y)
- Node I: (Y, W)
- Node J: (Z)
Binary Code Instrumentation

• Modified x86 Detours [USENIX WinNT ‘99]
  • Arbitrary instruction instrumentation

• NOP insertion using BISTRO [ESORICS ‘13]
  • Original program state preserved

```c
* PG_for_X (PC, SP) {
*   <Save Registers>
*   <Set PC and SP>
*   <Save Error Number>
*   // Do Performance Check
*   <Save Error Number>
*   <Restore Registers>
*   return
* }
```

```c
Foo (...) {
  ...
  + CALL PG_for_X
  Instruction X
  ...
}
```
### Evaluation

- Diagnosis of real-world performance bugs

<table>
<thead>
<tr>
<th>Program Name</th>
<th>Bug ID</th>
<th>Root Cause Binary</th>
<th>Unit Call Trees</th>
<th>Unit Call Paths</th>
<th>Unit Functions</th>
<th>Inserted Unit Perf. Guards</th>
<th>Unit Thresh-hold (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>45464</td>
<td>Internal Library</td>
<td>8</td>
<td>17,423</td>
<td>635</td>
<td>138</td>
<td>9,944</td>
</tr>
<tr>
<td>MySQL Client</td>
<td>15811</td>
<td>Main Binary</td>
<td>24</td>
<td>255,126</td>
<td>106</td>
<td>13</td>
<td>997</td>
</tr>
<tr>
<td>MySQL Server</td>
<td>49491</td>
<td>Main Binary</td>
<td>8</td>
<td>270,454</td>
<td>980</td>
<td>303</td>
<td>2,079</td>
</tr>
<tr>
<td>7-Zip File Manager</td>
<td>S4</td>
<td>Main Binary</td>
<td>6</td>
<td>26,842</td>
<td>143</td>
<td>120</td>
<td>101</td>
</tr>
<tr>
<td>Notepad++</td>
<td>2909745</td>
<td>Main Binary</td>
<td>16</td>
<td>352,831</td>
<td>711</td>
<td>370</td>
<td>6,797</td>
</tr>
<tr>
<td>ProcessHacker</td>
<td>3744</td>
<td>Main Binary</td>
<td>1</td>
<td>47,910</td>
<td>86</td>
<td>23</td>
<td>3,104</td>
</tr>
<tr>
<td>ProcessHacker</td>
<td>5424</td>
<td>Plug-in</td>
<td>32</td>
<td>62,136</td>
<td>69</td>
<td>19</td>
<td>10</td>
</tr>
</tbody>
</table>

- 1-32 Distinct Call Trees, 4,000-352,000 Call Paths, and 65-980 Functions
- Average of: 124 Insertions, and 2,337 ms Per Buggy Unit
Use Case: Unit Call Stack Traces

• Case 1: Apache HTTP Server

<table>
<thead>
<tr>
<th>Unit Call Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>T libapr-1.dll!convert_prot</td>
</tr>
<tr>
<td>- libapr-1.dll!more_finfo</td>
</tr>
<tr>
<td>- libapr-1.dll!apr_file_info_get</td>
</tr>
<tr>
<td>- libapr-1.dll!resolve_ident</td>
</tr>
<tr>
<td>R libapr-1.dll!apr_stat</td>
</tr>
<tr>
<td>- mod_dav_fs.so!dav_fs_walker</td>
</tr>
<tr>
<td>- mod_dav_fs.so!dav_fs_internal_walk</td>
</tr>
<tr>
<td>- mod_dav_fs.so!dav_fs_walk</td>
</tr>
<tr>
<td>- ...</td>
</tr>
<tr>
<td>- libhttpd.dll!ap_run_process_connection</td>
</tr>
<tr>
<td>- libhttpd.dll!ap_process_connection</td>
</tr>
<tr>
<td>- libhttpd.dll!worker_main</td>
</tr>
</tbody>
</table>

Performance Bug: Apache 45464

• Case 2: 7-Zip File Manager

<table>
<thead>
<tr>
<th>Unit Call Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>T 7zFM.exe!NWindows::NCOM::MyPropVariantClear</td>
</tr>
<tr>
<td>R 7zFM.exe!Refresh_StatusBar</td>
</tr>
<tr>
<td>- 7zFM.exe!OnMessage</td>
</tr>
<tr>
<td>- 7zFM.exe!NWindows::NControl::WindowProcedure</td>
</tr>
<tr>
<td>- USER32.dll!InternalCallWinProc</td>
</tr>
<tr>
<td>- ...</td>
</tr>
<tr>
<td>- USER32.dll!DispatchMessageWorker</td>
</tr>
<tr>
<td>- USER32.dll!DispatchMessageW</td>
</tr>
<tr>
<td>- 7zFM.exe!WinMain</td>
</tr>
</tbody>
</table>

Root Cause Functions

Performance Bug: 7-Zip S3
Performance Overhead

- ApacheBench & SysBench: Overhead < 3%

- Apache HTTP Server

- MySQL Server
Related Works


Conclusion

• PerfGuard enables diagnosis of performance problems without source code and development knowledge.

• Unit-based performance profiling allows targeting a general scope of software.

• Automatically detects performance problems with low run-time overhead (< 3%).
Thank you!

Questions?

Chung Hwan Kim
chungkim@cs.purdue.edu