Benchmarking - What can it buy you?

http://www.ices.cmu.edu/ballista

John P. DeVale
devale@cmu.edu - (412) 268-4264 - http://www.ece.cmu.edu/~jdevale
Overview: Benchmarking – What can it buy you?

◆ General overview of Ballista

◆ What can you buy with the result
  • Improved robustness

◆ How much does it cost
  • Performance penalty?

◆ Conclusions
  • High robustness with Low performance penalty

A Ballista is an ancient siege weapon for hurling objects at fortified defenses.
Overview
System Robustness -- Improves Dependability

- **Graceful behavior in the presence of exceptional conditions**
  - Unexpected operating conditions
  - Activation of latent design defects

- **Robustness definition also includes operation in overloads**
  - Currently building mechanisms to perform this testing and analyze results

- **Research Goal**
  - *Metric for comparative evaluation of software robustness*
  - *Ability to apply metric results in a consistent fashion to improve robustness*
  - *Structure exception handling code to specifically leverage hardware performance features*
Ballista Software Testing Heritage

- **SW Testing requires:**
  - Test case
  - Module under test
  - *Oracle* (a “specification”)

- **Ballista uses:**
  - “Bad” value combinations
  - Module under Test
  - *Watchdog timer/core dumps*

---

**Ballista combines:**
- Domain testing ideas / Syntax testing ideas
- In general, “dirty” testing
**Ballista: Test Generation**

**API**

**TESTING OBJECTS**

**ORTHOGONAL PROPERTIES**

**TEST VALUES**

<table>
<thead>
<tr>
<th>File State</th>
<th>Buffer Type</th>
<th>Flags</th>
<th>IntValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPN_READ</td>
<td>MAPPED</td>
<td>STRING</td>
<td>MAXINT</td>
</tr>
<tr>
<td>OPN_WRITE</td>
<td>BUFFERED</td>
<td>READ</td>
<td>MININT</td>
</tr>
<tr>
<td>OPN_RW</td>
<td>NON_BUFFERED</td>
<td>WRITE</td>
<td>ZERO</td>
</tr>
<tr>
<td>CLOSED</td>
<td></td>
<td>APPEND</td>
<td>ONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LINE</td>
<td>NEGONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHARE</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PUBLIC</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MALLOC</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STATIC</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IOCHECK</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BUFCONST</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WHOLE</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MALLOC_STATIC</td>
<td></td>
</tr>
</tbody>
</table>

```c
Sfseek (Sfio_t *theFile=( Composite Value), int pos=0)
```
CRASH Severity Scale

◆ Catastrophic
  • Test computer crashes (both Benchmark and Starter abort or hang)
  • Irix 6.2: `munmap(malloc((1<<30)+1), ((1<<31)-1));`

◆ Restart
  • Benchmark process hangs, requiring restart

◆ Abort
  • Benchmark process aborts (e.g., “core dump”)

◆ Silent
  • No error code generated, when one should have been
    (e.g., de-referencing null pointer produces no error)

◆ Hindering
  • Incorrect error code generated
Where we currently are

- **Applied methodology across a wide range of software systems**
  - Operating Systems
  - User level libraries
  - DOD distributed simulation framework
  - Commercial Java Beans
  - Corporate COM/DCOM distributed control framework
  - Critical Military Systems

- **Improved testing granularity by decomposing data types into orthogonal properties**

- **Adding stress testing and resource overload capabilities**
What can we buy with it?

Can we get excellent robustness without sacrificing performance?
SFIO[korn91] – a brief introduction

◆ Idea:
  1. Measure something that is supposed to be bulletproof
  2. See if being really “bulletproof” of necessity costs performance

◆ The Safe, Fast, I/O library
  • Written by Korn and Vo at AT&T research, 1991
  • Addresses the many safety/robustness/reliability issues found in the Standard IO libraries

◆ Their goal: safe operation with robust exception handling without paying a performance premium

SFIO, the original version (1990)

- They couldn’t measure; but we can
- Up to 10x Improvements in robustness
- Low performance impact
So why bother with the metric?

◆ The authors of SFIO had no metric

◆ They fixed a large number of problems
  • BUT, they didn’t find them all!

◆ * The lack of quantitative feedback made it difficult to know how well they had done, and what their effort had gained them
  • And, the stuff they missed was in fact “obvious”
Our version 5-7x more robust

- The use of a metric – in our case Ballista – allowed us to improve performance with respect to exception handling an additional 5-7x

<table>
<thead>
<tr>
<th>Function</th>
<th>% Abort Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>open</td>
<td>STDIO: 1.14</td>
</tr>
<tr>
<td>write</td>
<td>STDIO: 5.99</td>
</tr>
<tr>
<td>read</td>
<td>STDIO: 9.42</td>
</tr>
<tr>
<td>close</td>
<td>STDIO: 2.00</td>
</tr>
<tr>
<td>fileno</td>
<td>STDIO: 7.80</td>
</tr>
<tr>
<td>seek</td>
<td>STDIO: 8.00</td>
</tr>
<tr>
<td>sfputc</td>
<td>STDIO: 0.00</td>
</tr>
<tr>
<td>sfgetc</td>
<td>STDIO: 0.00</td>
</tr>
</tbody>
</table>
Using a Metric leads to better results

- The result of no good metric?
  - They missed opportunities for easy robustness gains
  - They honestly thought they had found all the easy stuff

- The types of failures exhibited can be broadly classified as:
  - File permissions
  - Memory validation
CPU Cycles – wither thou goest?

- Better exception handling, but at what cost? – Not much <1%

Elapsed Time
File sizes 2x-8x larger for the axp (ALPHA) system

![Bar chart showing elapsed time for different benchmarks and file systems.]

- x86-STDIO
- x86-Original SFIO
- x86-Robust SFIO
- axp-STDIO
- axp-Original SFIO
- axp-Robust SFIO
Summary

- Without a good metric, even the best effort is just a stab in the dark
  - In this case, the metric was used as feedback to improve SW

- With a good metric we can do a better job with robustness, and know where to expend effort and what that effort buys us
  - Oh btw, the performance cost is not large (less than 1%)

- Ballista continues to evolve to address a wider domain of dependability issues
  - The “bug report” extraction feature helps with performance tuning as well