Predicting Faults from Cached History

A. Kim  T. Zimmermann  J. Whitehead  A. Zeller

Presented by Lukas Gmüör

11. May 2010
Intent of Paper

• Empirical evidence of fault localities
• Accurate prediction of faults
• Adaptive prediction of faults
Why should it work?
Faults do not occur uniformly at random in time or space
Bug localities

Where do we find bugs?
Bug localities

Where do we find bugs?

• Entity has been added recently: «new-entity locality»
Bug localities

Where do we find bugs?

• Entity has been added recently: «new-entity locality»
• Entity was changed recently: «changed-entity locality»
Bug localities

Where do we find bugs?

• Entity has been added recently: «new-entity locality»
• Entity was changed recently: «changed-entity locality»
• Entity has introduced a fault recently:
  – «Temporal locality»
  – «Spatial locality»
Temporal locality:

• Multiple faults within the same entity
• If there were no faults in a long time it is less likely to have faults
Temporal vs Spatial locality

Spatial locality:
• When an entity has a fault there is a good chance of other nearby entities having faults
  – Physical distance: Same file/directory
  – Logical coupling: entities are changed together:
    • $distance(e_1, e_2) = \begin{cases} \frac{1}{\text{count}(e_1, e_2)}, & \text{count}(e_1, e_2) > 0 \\ \infty, & \text{otherwise} \end{cases}$
    • $\text{count}(e_1, e_2) = \text{number of times } e_1 \text{ and } e_2 \text{ have been changed together}$
Cache

- Well known subject from Computer Architecture
- Goal: Very small size and yet high hit rate
- Cache from this paper:
  - Cache size = 10% of total files/methods
  - Different replacement strategies
  - Varying cacheline sizes
  - Different maximum cacheline size
  - Two different caches
  - Avoid cold cache at beginning
  - Cache is initialized with entities likely to have faults (greatest line of code)
  - Prefetching
Replacement strategies

• Least recently used («LRU»)
• LRU weighted by number of changes
• LRU weighted by number of previous faults («BUG»)
• Different bug localities enforce different Cache operations:
  – Prefetch(changed-entity and new-entity locality)
  – Normal (temporal and spatial locality)
• Maximum prefetch size is a constant percent of cache size
Why two caches?

• One cache for empirical evidence
• One cache for future prediction
Bugcache vs Fixcache

**Bugcache**
- Empirical evidence
- Updates when fault was introduced

**Fixcache**
- Fault prediction
- Updates when fault is fixed
Bugcache vs Fixcache

**Bugcache**

1. Check in cache?

2. If miss update Cache

$t_{bug}$

**Fixcache**

1. Check in cache?

2. Identify bug introducing Change

$t_{bug}$

3. If miss update Cache, with localities at $t_{bug}$

$t_{fix}$
Brute force approach:
• All replacements strategies
• Different maximum cacheline sizes
• Different prefetch sizes

Why?
• Transaction recovery over the product not only the file level
  – SVN: Ok
  – CVS: grouped to a transaction using a sliding window approach
• Finding bug-introducing changes/fixes
  – Keywords «bug» and «fix»
  – References to bug reports
<table>
<thead>
<tr>
<th>Project</th>
<th>Language</th>
<th>SCM</th>
<th>Revisions</th>
<th>Entities</th>
<th>Files</th>
<th>Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache 1.3</td>
<td>C</td>
<td>SVN</td>
<td>7747</td>
<td>2113</td>
<td>154</td>
<td>1954</td>
</tr>
<tr>
<td>Subversion</td>
<td>C</td>
<td>SVN</td>
<td>6029</td>
<td>3693</td>
<td>255</td>
<td>1566</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>C</td>
<td>CVS</td>
<td>14650</td>
<td>8659</td>
<td>598</td>
<td>19902</td>
</tr>
<tr>
<td>Mozilla*</td>
<td>C++</td>
<td>CVS</td>
<td>109636</td>
<td>8203</td>
<td>396</td>
<td>52265</td>
</tr>
<tr>
<td>Jedit</td>
<td>Java</td>
<td>CVS</td>
<td>1386</td>
<td>5429</td>
<td>420</td>
<td>3060</td>
</tr>
<tr>
<td>Columba</td>
<td>Java</td>
<td>CVS</td>
<td>2848</td>
<td>8428</td>
<td>1428</td>
<td>720</td>
</tr>
<tr>
<td>Eclipse*</td>
<td>Java</td>
<td>CVS</td>
<td>78948</td>
<td>33214</td>
<td>3330</td>
<td>15217</td>
</tr>
</tbody>
</table>
Bugcache Evaluation – File level

1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0

Apache  Subversion  PostgreSQL  Mozilla  Jedit  Columba  Eclipse

20
Bug locality – Method level

Apache 1.3

- Temporal locality
- Spatial locality
- Initial prefetch
- Changed-entity locality
- New-entity locality
Fixcache Evaluation – File level

![Bar Chart]

- Apache
- Subversion
- PostgreSQL
- Mozilla
- Jedit
- Columba
- Eclipse

<table>
<thead>
<tr>
<th>Software</th>
<th>Bugcache</th>
<th>Fixcache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>0.80</td>
<td>0.79</td>
</tr>
<tr>
<td>Subversion</td>
<td>0.73</td>
<td>0.79</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>0.81</td>
<td>0.87</td>
</tr>
<tr>
<td>Mozilla</td>
<td>0.91</td>
<td>0.93</td>
</tr>
<tr>
<td>Jedit</td>
<td>0.86</td>
<td>0.92</td>
</tr>
<tr>
<td>Columba</td>
<td>0.82</td>
<td>0.84</td>
</tr>
<tr>
<td>Eclipse</td>
<td>0.93</td>
<td>0.93</td>
</tr>
</tbody>
</table>

Bugcache vs. Fixcache

- Diagram shows a comparison of Bugcache and Fixcache for different software projects.
Fixcache Evaluation – Method level

![Bar chart showing comparison between Bugcache and Fixcache across various software projects.](image-url)
Conclusion

• Results are equal or better than previous prediction models

• Empirical evidence of fault localities
• Accurate prediction of faults
• Adaptive prediction of faults
My opinion

• The authors call it adaptive...

• Cache should have been further away from a processor cache (check if removed entity from cache is more likely to have a fault than inserted entity)

• Keep a statistic about misses and adapt parameters automatically to reduce them
Threads to Validity

- Only 7 projects
- Only open source projects
- Data collection is bad ($\approx 50\%$)
- Rename interpreted as delete and new
Optimal strategy

- LRU weighted by number of previous faults
- Block size of 30-50%
- Pre-fetch size of 10-30%
# Optimal strategy - Bugcache

<table>
<thead>
<tr>
<th>Application</th>
<th>Block size</th>
<th>Prefetch size</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache f*</td>
<td>7</td>
<td>0</td>
<td>LRU</td>
</tr>
<tr>
<td>Apache m*</td>
<td>127</td>
<td>24</td>
<td>BUG</td>
</tr>
<tr>
<td>Subversion f</td>
<td>6</td>
<td>0</td>
<td>LRU</td>
</tr>
<tr>
<td>Subversion m</td>
<td>366</td>
<td>54</td>
<td>BUG</td>
</tr>
<tr>
<td>PostgreeSQL f</td>
<td>1</td>
<td>0</td>
<td>LRU</td>
</tr>
<tr>
<td>PostgreeSQL m</td>
<td>347</td>
<td>102</td>
<td>BUG</td>
</tr>
<tr>
<td>Mozilla f</td>
<td>17</td>
<td>0</td>
<td>LRU</td>
</tr>
<tr>
<td>Mozilla m</td>
<td>641</td>
<td>96</td>
<td>LRU</td>
</tr>
<tr>
<td>Jedit f</td>
<td>17</td>
<td>0</td>
<td>LRU</td>
</tr>
<tr>
<td>Jedit m</td>
<td>1</td>
<td>42</td>
<td>BUG</td>
</tr>
<tr>
<td>Columba f</td>
<td>46</td>
<td>0</td>
<td>BUG</td>
</tr>
<tr>
<td>Columba m</td>
<td>641</td>
<td>132</td>
<td>BUG</td>
</tr>
<tr>
<td>Eclipse f</td>
<td>331</td>
<td>0</td>
<td>BUG</td>
</tr>
<tr>
<td>Eclipse m</td>
<td>665</td>
<td>0</td>
<td>BUG</td>
</tr>
</tbody>
</table>
Benefits of fault prediction

• Cache can serve as a priority list to test software
• Can help Developer during development