MRI: Comparison-based Diagnosis for Performance Bugs

Linhai Song

Professor: Shan Lu

What are *Performance Bugs?*

- Software defects
  - Heavy performance impact
  - Simple patches
- Performance bugs inevitably escape into deployed software

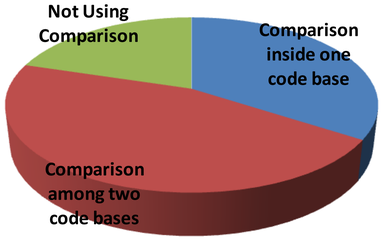
What are Diagnosis Tools?

- Tools
  - Fault localization
  - Give out fix suggestions
- In-house diagnosis tools
- On-line diagnosis tools
- Little support for diagnosing perf. bugs

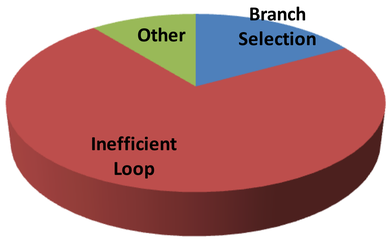
Characteristics Study

- Study 65 user perceivable performance bugs
- From Apache, Chrome, Mozilla, MySQL, GCC

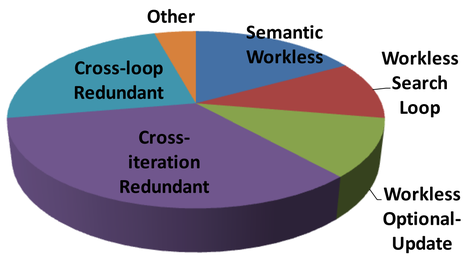
How Users Notice Performance Bugs



What Caused Performance Loss



What Caused Inefficient Loops



MRI Design And Implementation

Branch Collection

Overview

- Helpful for branch related bugs
- Helpful for loop related bugs
- Pin for iMRI (in-house diagnosis)
- LBR for oMRI (on-line diagnosis)

How to get not-taken events from LBR sampling reports?

Sample codes:

```
int iResult = 0 ;
for( int i = 0; i < 100; i ++ )
{
    if( i % 10  == 8 )
    {
        iResult ++;
    }
    if( i % 10 == 9)
    {
        function2();
    }
}
```

Branch Instructions:

```
40088c:   jmp    4008ea
4008be:   mov    ....
...
4008b4:   jne    4008ba
4008b6:   addl   ...
4008ba:   mov    ...
...
4008df :  jne    4008e6
4008e1:   callq  400854 <_Z9function2v>
...
4008ea:   cmpl   ...
...
4008f3:   jne    40088e
```

One LBR Sample:

```
4008f3  ----->  40088e
4008b4  ----->  4008ba
4008df  ----->  4008e6
4008f3  ----->  40088e
4008b4  ----->  4008ba
4008df  ----->  4008e6
4008f3  ----->  40088e
4008df  ----->  4008e6
4008f3  ----->  40088e
4008df  ----->  4008e6
4008b4  ----->  4008ba
4008e1  ----->  400854
```

Descriptions:
One LBR sample is ***16*** consecutive branch taken events.
We could get not-taken events between two neighboring taken events.

4008b4 is not taken

4008df is not taken

Continue to check branch instructions in function2()

Statistical Models

Overview

- Look for buggy branch and loops
- CBI [1] model for branches
- Delta-LDA [2] for loops

CBI Model

$$Failure(P) = \frac{F(P)}{S(P) + F(P)}$$

$$Context(P) = \frac{F(P\ observed)}{S(P\ observed) + F(P\ observed)}$$

$$Increase(P) = Failure(P) - Context(P)$$

$$Importance(P) = \frac{2}{\frac{1}{Increase(P)} + \frac{1}{\log(F(P))\ /\log(NumP)}}$$

Descriptions:
Use Importance(P) to rank taken and not-taken predicates.
Prefer branch executed in both good and bad runs, but only taken or not taken in bad runs.

Delta-LDA

- Evolve from LDA for debugging
- A branch ranked high by Delta-LDA
  - More likely to appear in bad runs
  - More frequent to appear in bad runs
- Cluster branches by innermost loops
- Utilize Delta-LDA to look for hot loops

Post-mortem Analysis

Overview

- 5 checkers for loop-related bugs
  - Static information from LLVM
  - Dynamic information from Pin or LBR

```
uint parent_tag(NODE *items,
                uint nitems, uint level) {
    for (p= last; p >= items; p--)
    {   // buggy search loop
        if (p->level == level &&
            p->type == NODE_TAG ) {
            return p - items;
        }
    }
}
```

MySQL27287 *Workless Search Loop*

Condition 1 : loop type is search loop
Condition 2 : iteration number is large

```
//warn_for_collisions is called inside a outer loop
void warn_for_collisions(…) {
    for(tmp=list; tmp; tmp= tmp->next) {
        if(tmp->writer)
            warn_for_collisions_1 ( …);
    }
}
```

Condition 1 : loop type is optional update loop
Condition 2 : working ratio is too low

```
find_alloca_call (…) {
-walk_tree ( exp, find_alloca_call_1,  NULL);
+   htab = htab_create(….);
+   walk_tree( exp, find_alloca_call_1, htab);
}
tree walk_tree ( tree * tp, walk_tree_fn func,
                 void * htab_ ) {
    for( i = 0; i < len; i ++ )
    {   //Buggy Loop
        result = walk_tree( TREE_OPERAND(*tp, i),
                            func, data, htab);
    }
}
```

GCC1687 *Cross-Iteration Redundancy*

Condition : function calls with same parameter

```
//insert is called in a outer loop
insert(int index, E element) {
    arraycopy(Data, index, Data,
              index+1, size - index);
    Data[index] = element;
} // add element in a sort array
static void arraycopy(…) {
    …
    while(count -- > 0) *to-- = * from--;
    …
}
```

Apache47223 *Incremental Update*

Cond. 1: Inner loop overwrites an array
Cond. 2: Outer loop doesn't use array value

```
//set_use_link is called inside a outer loop
static inline void set_use_link(…) {
    …
    while(prev->use != NULL || prev->stmt==NULL)
        prev = prev->prev; // inner loop
    …
}
```

GCC21430 *Reusable Computation*

Cond. 1: side effect depends on element and inner loop invariant variables
Cond. 2: Both of them not changed

Evaluation

- We evaluate MRI on 17 bugs
  - 15 C++ bugs and 2 Java bugs
  - 7 branch-related and 12 loop-related
- 2 bugs have false positives
- 1.8% - 9.0% for oMRI

BugID	KLOC	Categorization	FP
Mozilla258793	3482	Branch-Related	0
Mozilla299742	3482	Branch-Related	0
MySQL40337	1207	Branch-Related	1
MySQL42649	1164	Branch-Related	0
MySQL44723	1164	Branch-Related	0
GCC15209	2174	Branch-Related	0
Mozilla411722	126	Branch-Related	0
Mozilla347306	99	Workless Search Loop	0
MySQL27287	995	Workless Search Loop	0
GCC8805	2174	Workless Search Loop	0
GCC46401	5233	Workless Optional Update loop	3
MySQL38491	1207	Cross-Iteration Redundancy	0
GCC1687	1799	Cross-Iteration Redundancy	0
Apache34464	0.1	Cross-Loop Redundancy	0
Apache47223	0.2	Cross-Loop Redundancy	0
MySQL15811	1087	Semantic Workless	0
GCC21430	3002	Cross-Loop Redundancy	0

Table 1: Benchmark information and results from statistical models

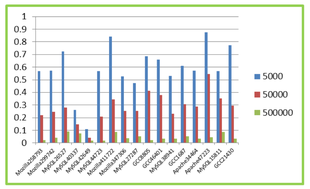


Figure 1: Overhead changed with different sample rate

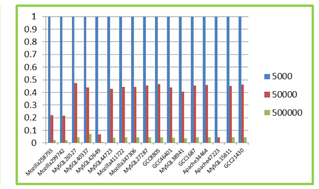


Figure 2: Samples changed with different sample rate

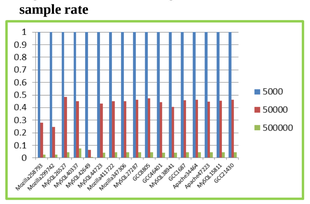


Figure 3: Taken events changed with different sample rate

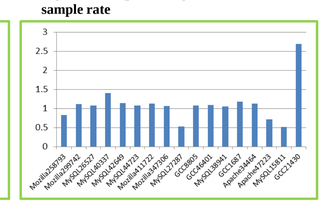


Figure 4: Compare taken/not-taken ratio from Pin and LBR

References:

- B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan. Scalable statistical bug isolation. In PLDI'05.
- D. Andrzejewski, A. Mulhern, B. Liblit, and X. Zhu. Statistical debugging using latent topic models. In ECML'2007.
- G. Jin, L. Song, X. Shi, J. Scherpelz, and S. Lu. Understanding and detecting real-world performance bugs. In PLDI'2012.