Performance Annotations for Complex Software Systems

Daniele Rogora*  Antonio Carzaniga*  Amer Diwan§  Matthias Hauswirth*
Robert Soulé†

*USI, Switzerland  †Yale University, USA  §Google, USA

EuroSys’20
Performance Analysis is Complex!
Algorithmic Performance Analysis

`std::list<int>::sort()`
std::list<int>::sort()

documented complexity:
\( time = O(n \log n) \)
Real Performance

std::list<int>::sort()

Time (seconds) vs. List size (million) chart showing the performance of `std::list<int>::sort()`.
std::list<int>::sort()
Performance Analysis with Performance Annotations

*Real, expected behavior as a function of input/state features*
Performance Analysis with *Performance Annotations*

Real, expected behavior as a function of input/state features
Performance Analysis with *Performance Annotations*

*Real, expected behavior as a function of input/state features*
Performance Analysis with *Performance Annotations*

- actual behavior
- concrete metrics
- specific characterization
- not merely an *aggregate* profile
- significant statistics

*Real, expected behavior as a function of input/state features*
Performance Analysis with *Performance Annotations*

**Actual behavior**
- Concrete metrics

**Expected behavior**
- As a function of input/state features

**Significant statistics**

Specific characterization
- Not merely an aggregate profile

For each module/function of interest:

\[
\text{metric}_i = f_i(\text{feature}, \ldots)
\]
Performance Analysis with Performance Annotations

Real, expected behavior as a function of input/state features

For each module/function of interest:

\[ \text{metric}_i = f_i(\text{feature}, \ldots) \]
Performance Analysis with *Performance Annotations*

Real, expected behavior as a function of input/state features

For each module/function of interest:

\[ \text{metric}_i = f_i(\text{feature}, \ldots) \]

- actual behavior
- concrete metrics
- specific characterization
  - not merely an *aggregate* profile
- significant statistics
- run-time
- memory allocation
- lock-holding time
  - ... input parameters, global variables,
  - ... even in nested, structured objects
  - *identified automatically!*

6/29
std::list<int>::sort.time(this) {
    uint s = *(this->_M_impl._M_node._M_storage._M_storage);

    [s > 49584 && s < 1450341]
    Norm(53350.31 - 2.10*s + 0.12*s*log(s), 12463.88);

    [s > 1589482 && s < 2085480]
    Norm(-90901042.29 + 63.11*s, 899547.29);

    [s > 2098759 && s < 3415880]
    Norm(56712024.50 + 35.38*s, 3379580.27);
}
std::list<int>::sort.time(this) {
    uint s = *(this->_M_impl._M_node._M_storage._M_storage);

    [s > 49584 && s < 1450341]
    Norm(53350.31 - 2.10*s + 0.12*s*log(s), 12463.88);

    [s > 1589482 && s < 2085480]
    Norm(-90901042.29 + 63.11*s, 899547.29);

    [s > 2098759 && s < 3415880]
    Norm(56712024.50 + 35.38*s, 3379580.27);
}
std::list<int>::sort.time(this) {
    uint s = *(this->_M_impl._M_node._M_storage._M_storage);

    [s > 49584 && s < 1450341]
    Norm(53350.31 - 2.10*s + 0.12*s*\log(s), 12463.88);

    [s > 1589482 && s < 2085480]
    Norm(-90901042.29 + 63.11*s, 899547.29);

    [s > 2098759 && s < 3415880]
    Norm(56712024.50 + 35.38*s, 3379580.27);
}

function of interest metric
Performance Annotations

```
std::list<int>::sort.time(this) {
  uint s = *(this->_M_impl._M_node._M_storage._M_storage);

  [s > 49584 && s < 1450341]
  Norm(53350.31 - 2.10*s + 0.12*s*log(s), 12463.88);

  [s > 1589482 && s < 2085480]
  Norm(-90901042.29 + 63.11*s, 899547.29);

  [s > 2098759 && s < 3415880]
  Norm(56712024.50 + 35.38*s, 3379580.27);
}
```
std::list<int>::sort.time(this) {
    uint s = *(this->_M_impl._M_node._M_storage._M_storage);

    [s > 49584 && s < 1450341]
    Norm(53350.31 - 2.10*s + 0.12*s*log(s), 12463.88);

    [s > 1589482 && s < 2085480]
    Norm(-90901042.29 + 63.11*s, 899547.29);

    [s > 2098759 && s < 3415880]
    Norm(56712024.50 + 35.38*s, 3379580.27);
}
std::list<int>::sort.time(this) {
    uint s = *(this->_M_impl._M_node._M_storage._M_storage);

    if (s > 49584 && s < 1450341) {
        Norm(53350.31 - 2.10*s + 0.12*s*log(s), 12463.88);
    }

    if (s > 1589482 && s < 2085480) {
        Norm(-90901042.29 + 63.11*s, 899547.29);
    }

    if (s > 2098759 && s < 3415880) {
        Norm(56712024.50 + 35.38*s, 3379580.27);
    }
}
```cpp
std::list<int>::sort.time(this) {
    uint s = *(this->_M_impl._M_node._M_storage._M_storage);
    [s > 49584 && s < 1450341]
    Norm(53350.31 - 2.10s + 0.12s*log(s), 12463.88);
    [s > 1589482 && s < 2085480]
    Norm(-90901042.29 + 63.11s, 899547.29);
    [s > 2098759 && s < 3415880]
    Norm(56712024.50 + 35.38s, 3379580.27);
}
```

function of interest metric

feature: $s$=list size

scope (1)

scope (2)

scope (3)
get_func_mm_tree(RANGE_OPT_PARAM *param,
    Item *pred,
    Item_func *cond_func,
    Item *val,
    bool inv);

get_func_mm_tree.time(cond_func) {
    uint ac = cond_func->arg_count;
    Norm(156569 - 269.041*ac + 0.414447*ac^2, 15781.22);
}
get_func_mm_tree(RANGE_OPT_PARAM *param,
    Item *pred,
    Item_func *cond_func,
    Item *val,
    bool inv);

item_func.h alone is 3885 lines!

def get_func_mm_tree.time(cond_func) {
    uint ac = cond_func->arg_count;
    Norm(156569 - 269.041*ac + 0.414447*ac^2, 15781.22);
}
Automatic Feature Discovery

```c
mysql_execute_command(THD *thd,
    bool first_level);

mysql_execute_command.time(thd) {
    uint len = thd->m_query_string.len;
    uint dvv = thd->variables.dynamic_variable_version;
    Norm(168.65 + 4.94*len + 1886.87*dvv, 2489.04);
}
```
Automatic Feature Discovery

```c
mysql_execute_command(THD *thd,
    bool first_level);

uint len = thd->m_query_string.len;
uint dvv = thd->variables.dynamic_variable_version;
Norm(168.65 + 4.94*len + 1886.87*dvv, 2489.04);
```

**unexpected feature!**

Thd.m_query_length

<table>
<thead>
<tr>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>60</td>
</tr>
<tr>
<td>80</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>120</td>
</tr>
<tr>
<td>140</td>
</tr>
<tr>
<td>160</td>
</tr>
</tbody>
</table>

```c
```

**struct traversal**
Uses of Performance Annotations

- **Documentation**
  - automatic creation
  - readable annotations and graphs for performance analyst
  - feature names as in the program

- **Annotations as performance assertions**
  - detecting performance anomalies and regressions

- **Prediction**
  - extrapolation to unobserved feature values
  - annotation composition: new code that uses annotated functions
Freud
our prototype for C/C++

- Function Name
- Program Binary
- Workload
- Text Annotation
- Graph Annotation
Freud

- Static analysis
- Feature discovery
- Feature-extraction code

function name

binary program

workload

annotations
Freud

- Static analysis
- Feature discovery
- Feature-extraction code

DWARF

function name
binary program
workload

CODE INFO

annotations
Freud

- Static analysis
- Feature discovery
- Feature-extraction code

- Dyn. instrumentation with Pin
  - PinTool using code from DWARF

Function name
Binary program
Workload
Annotations

DWARF
PIN
CODE/INFO
Freud

- Function name
- Binary program
- Workload

DWARF
- Static analysis
- Feature discovery
- Feature-extraction code

PIN
- Dyn. instrumentation with Pin
- PinTool using code from DWARF
- Run instrumented program

CODE INFO
Freud

- Function name
- Binary program
- Workload
- Annotations

**DWARF**
- Static analysis
- Feature discovery
- Feature-extraction code

**PIN**
- Dyn. instrumentation with *Pin*
- *PinTool* using code from DWARF
- Run instrumented program

**CODE INFO**

**LOGS**
Freud

- Function name
- Binary program
- Workload
- Annotations

**DWARF**
- Static analysis
- Feature discovery
- Feature-extraction code

**PIN**
- Dyn. instrumentation with Pin
- PinTool using code from DWARF
- Run instrumented program

**CODE INFO**

**LOGS**
- Offline statistical analysis
- Find regressions and clusters
- R for stats, gnuplot for graphs

**STATISTICS**
DWARF: Finding Features

- function name
- binary program
- info code
DWARF: Finding Features

- function name
- binary program
- info code

EXPLORE TREE

- Find function
  - Find global variables
  - Find parameters
DWARF: Finding Features

entry point for the analysis of the target function

function name

binary program

info code

Find function

Find global variables

Find parameters

EXPLORE TREE
DWARF: Finding Features

- function name
- binary program
- info code

EXPLORE TREE
- Find function
- Find global variables
- Find parameters

VARIABLES
all variables accessible by the target function

DWARF
DWARF: Finding Features

function name

binary program

info code

EXPLORE TREE

Find global variables

Find parameters

Build class graph

Find function

VARIABLES

DWARF
DWARF: Finding Features

- Function name
- Binary program
- Info code

Exploration Tree:
- Find function
- Find global variables
- Find parameters
- Build class graph

Variables

Class Graph
DWARF: Finding Features

EXPLORE TREE
- Find function
- Find global variables
- Find parameters
- Build class graph

EXPLORE VARIABLES
- Generate info
- Check possible dynamic types
- Generate code

function name
binary program
info code
DWARF: Finding Features

EXPLORE TREE

- Find function
- Find global variables
- Find parameters
- Build class graph

EXPLORE VARIABLES

- Generate info
- Check possible dynamic types
- Generate code

Determine all possible dynamic types for each statically defined variable

VARIABLES

CLASS GRAPH

function name

binary program

info code
DWARF: Finding Features

- function name
- binary program
- info code

Find location:
- register
- address
  - absolute
  - offset from register

Explore complex types:
- find names
- find types

Find global variables
Find parameters
Check possible dynamic types
Generate info
Generate code
DWARF: Finding Features

- DW ARF: Finding Features

- function name
- binary program
- info code

- Find function
- Find global variables
- Find parameters

- Explore complex types:
  - find basic fields
  - find offsets within complex types
  - generate C code to read data
    - Pin_SafeCopy

- Generate info
- Generate code
- Check possible dynamic types
Evaluation
Evaluation

- Does *Freud* Produce Correct Information?
  - set of basic functions using that use sleep to exhibit a known performance

- Does *Freud* help understanding performance?
  - real world experiments with complex PHP and C++ software

- Does *Freud* find performance bugs?
  - real world experiments with performance bugs from the MySQL bugtracker
Does *Freud* Produce Correct Information?

**Quadratic**

```c
void __attribute__ ((noinline)) test_quad_int(int t) {
    for (int i = 0; i < t; i++) {
        usleep(t);
    }
}
```
Does Freud Produce Correct Information?

Quadratic

void __attribute__((noinline)) test_quad_int(int t) {
    for (int i = 0; i < t; i++) {
        usleep(t);
    }
}

test_quad_int(t).time {
    Norm(3657.73 + 1.74*t^2, 19.31);
}
Does Freud Produce Correct Information?

**Branches**

```c
void __attribute__ ((noinline)) test_linear_branches_one_f(int a, int b, int c) {
    if (a < 10) {
        for (int i = 0; i < 10 - a; i++) { usleep(400); }
    } else {
        usleep(4000);
        for (int i = 0; i < a - 10; i++) usleep(400);
    }
}
```

![Graph showing time in milliseconds vs. a]

```c
test_linear_branches_one_f(a).time {
    [a <= 9]
    Norm(6472.36 - 651.01*a, 46.55);
    [a > 9]
    Norm(-1613.27 + 638.57*a, 32.88);
}
```
Does *Freud* Produce Correct Information?

**Interaction Terms**

```c
void __attribute__((noinline)) test_interaction_linear_quad(int a, int b) {
    for (int i = 0; i < a; i++)
        usleep(b*b);
}
```

```
\[
\text{test_interaction}(a,b).\text{time} \{ \\
\quad \text{Norm}(69.51 + 75.26 \times a - 0.39 \times b^2 + 1.54\times a\times b^2, 11.69); \\
\}
\]
```
Evaluation

- Does *Freud* Produce Correct Information?
  - set of basic functions using that use sleep to exhibit a known performance

- Does *Freud* help understanding performance?
  - real world experiments with complex Php and C++ software

- Does *Freud* find performance bugs?
  - real world experiments with performance bugs from the MySQL bugtracker
Does *Freud* Help Understanding?

\[
\text{ff}_h2645\_extract\_rbsp.\text{time}(\text{length}, \text{cpu}\_\text{clock}) \{
\text{uint } l = \text{length};
\text{uint } clock = \text{cpu}\_\text{clock};
\text{Norm}(43.32 + 0.055*l - 1.46e-05*clock \\
- 1.75e-08*l*\text{clock}, 4.56);
\}
\]
Does *Freud* Work with Complex Cases?

\[ x_{264\_8\_encoder\_encode}.\text{wait\_time}(h, \text{pic\_in}) \{
\]

- bool sliced = h->param.b_sliced_threads;
- uint height = h->param.i_height;
- uint threads = h->param.i_threads;
- uint dequant = h->thread.dequant4_mf;
- bool det = pic_in->param.b_deterministic;

[sliced]
\[
\text{Norm}(-56362 + 189.17 \times \text{height} - 3221.21 \times \text{threads} - 1378.66 \times \text{dequant} - 152.83 \times \text{height}\times\text{det} - 6.48 \times \text{height}\times\text{threads} + 10044\times\text{threads}\times\text{det}, 1.05e+05 )
\]

[!sliced]
\[
0.55\text{Norm}(108.7, 188.65); 0.30\text{Norm}(7282, 51465.24); \ldots
\]

\}
Evaluation

- Does *Freud* Produce Correct Information?
  - set of basic functions using that use sleep to exhibit a known performance

- Does *Freud* help understanding performance?
  - real world experiments with complex Php and C++ software

- Does *Freud* find performance bugs?
  - real world experiments with performance bugs from the MySQL bugtracker
Does *Freud* Find Performance Regressions?

---

**Bug #92979**

MySQL 8.0 performance degradation on INSERT with foreign_key_checks=0

<table>
<thead>
<tr>
<th>Submitted:</th>
<th>28 Oct 2018 13:51</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reporter:</td>
<td>Predrag Zivanovic</td>
</tr>
<tr>
<td>Status:</td>
<td>Verified</td>
</tr>
<tr>
<td>Category:</td>
<td>MySQL Server: InnoDB storage engine</td>
</tr>
<tr>
<td>Version:</td>
<td>8.0.13 Community Server</td>
</tr>
<tr>
<td>Assigned to:</td>
<td>CPU Architecture: x86</td>
</tr>
<tr>
<td>Tags:</td>
<td>dump, foreign keys</td>
</tr>
</tbody>
</table>


**Description:**
There is significant performance degradation between MySQL 5.7 and MySQL 8.0 when importing SQL dump with foreign keys and with foreign_key_checks=0. It looks like MySQL 8.0 is checking foreign keys references even with foreign_key_checks=0, only without error message.

**How to repeat:**
Here is MySQL dump file attached. On new fresh installation of MySQL 5.7 it took 15 seconds to import ... on MySQL 8.0 it took more then 400 seconds. InnoDB storage engine, default settings in both cases.
Does Freud Find Performance Regressions?

### 5.7.24

```c
mysql_execute_command(thd).time{
    uint len = thd->m_query_string.len;
    Norm(6630.19 + 0.86*len, 15.78);
}
```

### 8.0.11

```c
mysql_execute_command(thd).time{
    uint len = thd->m_query_string.len;
    Norm(168.65 + 4.94*len + 1886.87*dvv, 2489.04);
}
```

**Description:**
Query optimization takes a very long time for a SELECT query on a composite index with a large list of tuples. The performance degradation as the list of tuples grows seems to be geometric, compared to linear performance of an unindexed query or one using simple AND/OR clauses.

My expectation is that performance of the IN() function using an index would be similar, if not better, than alternatives, and that query optimization would not take more time than query execution.

I believe this is an issue with the optimizer, as the use of the index even affects "EXPLAIN SELECT ..." queries.

**How to repeat:**
Does *Freud* Help Finding Bugs?

```
... IN,OR/AND
   IN,OR/AND OR/AND
   test_quick_select(...) get_mm_tree(...)
     IN
     get_func_mm_tree(...)
      IN IN IN
      get_mm_parts(...) tree_and(...) tree_or(...)
       IN
        IN
        key_or(...)
```
Does *Freud* Help Finding Bugs?
Does *Freud* Help Finding Bugs?

test_quick_select (THD *thd, Key_map keys_to_use, table_map prev_tables, ha_rows limit, bool force_quick_range, const enum_order interesting_order, const QEP_shared_owner *tab, Item *cond, Key_map *needed_reg, QUICK_SELECT_I **quick, bool ignore_table_scan);

test_quick_select.time (thd, cond) {
    uint len = thd->m_query_string.len;
    uint vptr = cond->_vptr.Parse_tree_nodetmpl;
    [vptr <= 562874922]
    Norm(467533 - 50.21*len + 0.0036*len^2, 282711.59);
    [vptr > 562874922]
    Norm(-53.603 + 0.057*len, 157.57);
}
Does *Freud* Help Finding Bugs?

```
IN, AND  
  ↓
  test_quick_select(...)  
  IN, AND  
  |   
  |   ↓
  |   get_mm_tree(...)  
  |   ↓
  |   IN
  |   ↓
  |   get_func_mm_tree(...)  
  |   ↓
  |   IN
  |   ↓
  |   get_mm_parts(...)  
  |   ↓
  |   IN
  |   ↓
  |   tree_and(...)  
  |   ↓
  |   IN
  |   ↓
  |   tree_or(...)  
  |   ↓
  |   IN
  |   ↓
  |   key_or(...)  
```
Does *Freud* Help Finding Bugs?

```
...  
IN, AND  
  
  
  test_quick_select(...)  
    IN, AND  
    AND  
    
    get_mm_tree(...)  
      IN  
      
      get_func_mm_tree(...)  
        IN  
        IN  
        IN  
        
        get_mm_parts(...)  
        tree_and(...)  
        tree_or(...)  
      
      key_or(...)  
```
Does *Freud* Help Finding Bugs?

```c
key_or(RANGE_OPT_PARAM *param, SEL_ROOT *key1, SEL_ROOT *key2);

key_or.time(key2) {
    uint e = key2->elements;
    Norm(-0.276 + 0.073*e + 0.062*e*log(e), 2.24);
}
```
**Performance Annotations**

- probabilistic representation of expected performance
- account for different modalities in the behavior
Conclusion

- **Performance Annotations**
  - probabilistic representation of expected performance
  - account for different modalities in the behavior

- **Freud**
  - automatically creates *performance annotations* for C/C++ programs
  - [https://github.com/usi-systems/freud](https://github.com/usi-systems/freud)
Conclusion

- **Performance Annotations**
  - probabilistic representation of expected performance
  - account for different modalities in the behavior

- **Freud**
  - automatically creates *performance annotations* for C/C++ programs
  - [https://github.com/usi-systems/freud](https://github.com/usi-systems/freud)

- We shown that performance annotations can be used in different real world cases
  - documentation
  - performance assertions
  - a tool to find performance bugs
**Conclusion**

- **Performance Annotations**
  - probabilistic representation of expected performance
  - account for different modalities in the behavior

- **Freud**
  - automatically creates *performance annotations* for C/C++ programs
  - [https://github.com/usi-systems/freud](https://github.com/usi-systems/freud)

- We shown that performance annotations can be used in different real world cases
  - documentation
  - performance assertions
  - a tool to find performance bugs

- Future work
  - prediction
  - composition
Performance Annotations for Complex Software Systems

Daniele Rogora*  Antonio Carzaniga*  Amer Diwan$  Matthias Hauswirth*
Robert Soulé†

*USI, Switzerland  †Yale University, USA  $Google, USA

EuroSys’20