Automated Reasoning and Detection of Specious Configuration in Large Systems with Symbolic Execution

Yigong Hu, Gongqi Huang, Peng Huang
Johns Hopkins University

OSDI 2020
Setting Configuration Is Difficult

Logging

- datadir = /var/lib/mysql
- relay_log = mysql-relay-bin
- relay_log_index = mysql-relay-index

- log = mysql-gen.log
- log_error = mysql-error.err
- log_warnings
- log_bin = mysql-bin
- log_sLOW_queries = mysql-slow.log
- logQueries_not_using_indexes
- log_query_time = 10 #default: 10
- max_binlog_size = 256M #max size for binlog before rolling
- expire_logs_days = 4 #binlog files older than this will be purged

- log_bin = mysql-bin
- log_sLOW_queries = mysql-slow.log
- logQueries_not_using_indexes
- log_query_time = 10 #default: 10
- max_binlog_size = 256M #max size for binlog before rolling
- expire_logs_days = 4 #binlog files older than this will be purged

Buffer

- thread_stack = 256K #default: 32K, 64bit: 256K
- sort_buffer_size = 1M #default: 2M, larger may cause perf issues
- read_buffer_size = 1M #default: 128K, change in increments of 4K
- read_rnd_buffer_size = 1M #default: 256K
- join_buffer_size = 1M #default: 128K
- binlog_cache_size = 64K #default: 32K, size of buffer to hold TX queries
- total per-thread buffer memory usage: 8832000K = 8.625GB

Query Cache

- query_cache_size = 32M #global buffer
- query_cache_limit = 512K #max query result size to put in cache

Connections

- max_connections = 2000 #multiplier for memory usage via per-thread buffers
- max_connect_errors = 100 #default: 10
- concurrent_insert = 2 #default: 1, 2: enable insert for all instances
- connect_timeout = 30 #default: 5.1.22: 5, +5.1.22: 10
- max_allowed_packet = 128M #max size of incoming data to allow
Misconfiguration ≠ Invalid Configuration

• Misconfiguration detection (PeerPressure[OSDI’04], Pcheck[OSDI’16])
  o Invalid setting
  o Introduced by average users

• Many misconfiguration are valid setting
  o 46.3% ~ 61.9% of misconfigurations have perfectly legal parameters*
  o The effect are hard to predict even for experts
  o Cause severe performance issue in production

For simplicity, we call them specious configuration

*: An Empirical Study on Configuration Errors in Commercial and Open Source Systems. SOSP’11
An Example Specious-Configuration Incident

Bob’s website backend
An Example Specious-Configuration Incident
An Example Specious-Configuration Incident

Why does SQL query use a wrong query plan

This problem typically happens when the estimated cost of an index scan is too high and doesn't correctly reflect reality.

# QUERY TUNING

```
enable_bitmapscan = on
enable_hashagg = on
enable_hashjoin = on
enable_indexscan = on
enable_indexonlyscan = on
enable_material = on
enable_mergejoin = on
enable_nestloop = on
enable_parallel_append = on
enable_sort = on
```

# Planner Cost Constants

```
# measured on an arbitrary scale
seq_page_cost = 1.0
random_page_cost = 1.0
cpu_tuple_cost = 0.01
cpu_index_tuple_cost = 0.005
cpu_operator_cost = 0.0025
parallel_tuple_cost = 0.1
parallel_setup_cost = 1000.0
jit_above_cost = 100000
jit_inline_above_cost = 500000
jit_optimize_above_cost = 500000
min_parallel_table_scan_size = 8MB
min_parallel_index_scan_size = 512kB
effective_cache_size = 4GB
```
An Example Specious-Configuration Incident

```sql
...  
#QUERY TUNING
enable_bitmapscan = on
enable_hashagg = on
enable_hashjoin = on
enable_indexscan = on
enable_indexonlyscan = on
enable_material = on
enable_mergejoin = on
enable_nestloop = on
enable_parallel_append = on
enable_seqscan = on
enable_sort = on

# - Planner Cost Constants
seq_page_cost = 1.0  # measured on an arbitrary scale
random_page_cost = 1.0  # same scale as above

cpu_tuple_cost = 0.01  # same scale as above
cpu_index_tuple_cost = 0.005  # same scale as above
cpu_operator_cost = 0.0025  # same scale as above
parallel_tuple_cost = 0.1  # same scale as above
parallel_setup_cost = 1000.0  # same scale as above
jit_above_cost = 100000  # perform JIT compilation
jit_inline_above_cost = 500000  # inline small functions
jit_optimize_above_cost = 500000  # use expensive JIT optimizations
min_parallel_table_scan_size = 8MB
min_parallel_index_scan_size = 512kB
effective_cache_size = 4GB
...
```
Specious Configuration Is Prevalent

Today’s outage for several Google services

More Details on Today’s Outage
September 24, 2010 at 8:29 AM

Early today Facebook was down or unreachable for many of you for approximately 2.5 hours. This is the worst outage we’ve had in over four years, and we wanted to first of all apologize for it. We also wanted to provide much more technical detail on what happened and share.

Summary of the December 24, 2012 Amazon ELB Service Event in the U.S.

We would like to share more details with our customers about the event that occurred with the Amazon Elastic Load Balancing Service in the U.S. on December 24, 2012. While the service disruption only affected applications using the ELB service (and only a fraction of the ELB load balancers were affected), the impact for a prolonged period of time.

Google apologizes for service outage, reveals a 10% drop in YouTube views

A simple misconfiguration caused the outage

By Rounak Jain  June 5, 2019, 1:13 p.m.

And if config changes in cloud infrastructure systems contributing to incidents is your thing, we also have one from Google this week (@SREWeekly is the gift that keeps on giving):

status.cloud.google.com/incident/zall/...
What Is Missing From Current Tool?

- **Black-box testing is experimental**
  - Limited code coverage
  - Tailored to testing environment, specific configuration and input

- **Administrators have more questions:**
  - What happens if I change this setting from X to Y?
  - How would this setting perform with 100 nodes?
  - If my workload changes to mostly read-only, is this setting acceptable?
  - I plan to upgrade from HDD to SSD, should I update the config?
  - ...
To tackle specious configuration, we need an** analytical** approach to systematically **reason** about the performance effect of configuration.
Our Solution: Violet

S1: Explore performance effect with symbolic execution
   o Make configuration and input as one type of symbolic input
   o Symbolic explore the system code path with symbolic config & input
   o Derive performance impact model for each configuration

S2: Given concrete input, parameters, env info
   o Answer admins’ questions
   o Violet checker detects specious configuration based on the impact model
Outline

- Motivation
- Specious Configuration Code Patterns
- Violet Overview
- Evaluation
Code Pattern 1: Costly Operation

```c
int write_row() {
    if (autocommit) {
        ...
        trx_commit_complete();
    } else {
        trx_mark_sql_stat_end();
    }
}

uint trx_commit_complete() {
    if (flush_at_trx_commit==1) {
        log_group_write_buf();
        fil_flush();
    } else if (flush_at_trx_commit==2) {
        log_group_write_buf();
    } else {
        /* do nothing */
    }
}
```

• Some expensive operations is executed in one branch
Code Pattern 2: Additional Synchronization

```c
void mysql_parse(THD *thd) {
    if (send_result_to_client(thd) <= 0) {
        mysql_execute_command(thd);
    }
}

int mysql_execute_command(THD *thd) {
    case SQLCOM_SELECT:
        open_and_lock_tables(thd, all_tables);
        break;
    case SQLCOM_LOCK_TABLES:
        lock_tables_open_and_lock_tables(thd);
        if (query_cache_wlock_invalidate)
            invalidate_query_block_list();
    }

void invalidate_query_block_list() {
    free_query(list_root->block());
}
```

- Lead to additional table lock
Code Pattern 3: Slow Execution Flow

```c
void mysql_parse(THD *thd) {
    if (send_result_to_client(thd) <= 0) {
        mysql_execute_command(thd);
    }
}

int mysql_execute_command(THD *thd) {
    switch (SQLCOM_SELECT) {
    case SQLCOM_SELECT:
        open_and_lock_tables(thd, all_tables);
        break;
    case SQLCOM_LOCK_TABLES:
        lock_tables_open_and_lock_tables(thd);
        if (query_cache_wlock_invalidate)
            invalidate_query_block_list();
    }
    free_query(block());
}

• Lead to slow execution flow
```
Code Pattern 4: Frequent Crossing Threshold

```c
uint64_t log_reserve_and_open(uint len) {
    ...
    loop:
        ...
        if (len >= log->buf_size / 2) {
            log_buffer_extend((len + 1) * 2);
        }
        len_upper_limit = LOG_BUF_WRITE_MARGIN + (5 * len) / 4;
        if (log->buf_free + len_upper_limit > log->buf_size) {
            mutex_exit(&log->mutex);
            log_buffer_flush_to_disk();
            goto loop;
        }
    }

    • Costly operation being frequently triggered the costly operation
```
Static Analysis?

- **The four patterns are high-level characterizations**
  - Mapping them to specific code requires a lot of domain knowledge

- **Patterns are incomplete**
  - Other patterns and many variants

- **Fundamental limitations**
  - Infeasible paths
  - Performance is hard to be estimated statically
Parameter Affects Execution Flows

- A general characteristic is...
  - Different parameter causes different execution code path
  - Some path is extremely slower than others
  - Context-dependency

Detecting specious configuration = finding **slow** execution path + deducing triggering **condition**
Symbolic Execution

• Violet uses symbolic execution to find many slow paths and deduce their triggering conditions

• Advantages
  o Analyze system code without being limited by code patterns
  o Explored paths are feasible in native execution
  o Measure concrete performance from execution
Outline

- Motivation
- Specious Configuration Code Patterns
- Violet Overview
- Evaluation
Violet Overview

- System code
- Violet Hook
- Violent Hook
- Config dependency file
- Violet Tracer
- S2E
- Trace Analyzer
- Path Comparison
- Static analyzer

Configuration Constraints | Cost | Workload
--- | --- | ---
auto l=0 & flush == 1 | NULL | SQL => INSERT
auto l=0 & flush == 2 | NULL | SQL => INSERT
auto l=0 & flush l=2 | NULL | SQL => INSERT
auto l=0 | NULL | SQL != INSERT
How to Make Configuration Symbolic

• Making configuration file symbolic
  o Path explosion due to the parser

• Observation:
  o System usually keeps a dictionary to map configuration to variable
  o But we also need variable type, range and default value to make it symbolic

• Our approach:
  o Insert a hook to enumerate config variables and make them symbolic
Hooking API

- Insert after parse function
- Iterate all the variable
- Implement make_symbolic for each variable type

```c
static int get_options(int *argc_ptr, char **argv_ptr)
{
    my_init_dynamic_array(&all_options, sizeof(my_option));
    for (opt = my_long_options; opt < my_options_end; opt++) {
        insert_dynamic(&all_options, (uchar*) opt);
        ...
    }
    violet_make_mysql_options_symbolic();
    return 0;
}

void violet_make_mysql_options_symbolic()
{
    for (sys_var *var=all_sys_vars.first; var; var= var->next)
    if (is_config_in_targets(var->name.str))
        var->make_symbolic();
}
```
Violet Overview

System code

Violet Hook

Violet Overview
Static analyzer
Config dependency file

Violet Tracer
S2E

Trace Analyzer
Path Comparison

Configuration Constraints | Cost | Workload
---|---|---
auto l= 0 & flush == 1 | fill_flush | SQL = SQL INSERT
auto l= 0 & flush == 2 | NULL | SQL = SQL INSERT
auto l= 0 & flush l= 2 | NULL | SQL = SQL INSERT
auto == 0 | NULL | SQL l= SQL INSERT

24
Which Configuration to Make Symbolic?

• Making all configuration symbolic
  o Too many configurations -> path explosion
  o Many paths waste time on irrelevant execution
  o A lot of path constraints are misleading
Making Irrelevant Configuration Symbolic

```c
int write_row() {
    if (opt_c) {
        task1();
    } else {
        task2();
    }
    if (autocommit) {
        ...
        trx_commit_complete();
    } else {
        trx_mark_sql_stat_end();
    }
}
```

Constraints: `autocommit!=0&flush==1&opt_c==1`

Only making `related` configuration symbolic

- **Wasting long time to reach target configuration**
- **Optimal result**
- **Misleading result**

`opt_c` is irrelevant because it doesn’t impact the `autocommit`

```c
ulint trx_commit_complete() {
    if (flush_at_trx_commit==1) {
        log_group_write_buf();
        fil_flush();
    } else if 
        log_group_write_buf();
    } else {
        /* do nothing */
}
```
How to Find Related Configuration

• A related config is in **some execution flow** of target config

• Control dependency
  - X is control dependent on Y if X’s execution depends on a test at Y

```java
void main() {
    if (opty > 100)
        if (optx)
            init_x();
}
```

optx is control dependent on opty
optx, opty are related configurations
Relax Control Dependency

```c
int write_row() {
    if (autocommit) {
        ...
        if (opt_c)
            trx_commit_complete();
    } else {
        trx_mark_sql_stat_end();
    }
}

ulint trx_commit_complete() {
    if (flush_at_trx_commit==1) {
        log_group_write_buf();
        fil_flush();
    } else if (flush_at_trx_commit==2) {
        log_group_write_buf();
    } else {
        /* do nothing */
    }
}
```

- **flush** is related to **autocommit**
- **flush** is **not control dependent** on autocommit because `opt_c` is between **autocommit** and **flush**

Relaxing the definition to X’s execution depends on a test at Y and other parameters
Detecting Related Configuration

- Find enabler parameter set
- Find influenced parameter set
- Union both parameter set as related parameter

**Algorithm 1: Compute related parameters**

Func: GetRelatedConfigs

Input: $\mathcal{P}$: target program, $\mathcal{C}$: all parameter vars in $\mathcal{P}$

Output: $\mathcal{M}$: map from each parameter in $\mathcal{C}$ to the set of related parameters

$\mathcal{M} \leftarrow \{\}, \text{es\_map} \leftarrow \{\}, \text{ins\_map} \leftarrow \{\}$

foreach $p \in \mathcal{C}$ do

| $\text{es} \leftarrow \text{GetEnablerConfig}(p, \mathcal{P})$
| $\text{es\_map}[p] \leftarrow \text{es}$
| foreach $q \in \text{es}$ do
| $\text{ins\_map}[q] \leftarrow \text{ins\_map}[q] \cup \{p\}$
| foreach $p \in \mathcal{C}$ do
| $\mathcal{M}[p] \leftarrow \text{es\_map}[p] \cup \text{ins\_map}[p]$

return $\mathcal{M}$
Violet Overview

System code

Violet Hook

Static analyzer

Violet Tracer

S2E

Trace Analyzer

Path Comparison

Config dependency file

Configuration Constraints | Cost  | Workload
--- | --- | ---
auto l== 0 & flush == 1 | fil_flush | SQL => INSERT
auto l== 0 & flush == 2 | NULL | SQL => INSERT
auto l== 0 & flush l== 2 | NULL | SQL => INSERT
auto == 0 | NULL | SQL l== INSERT
Lightweight Symbolic Tracer

- Extensive profiling can incur too much overhead to the symbolic engine and cause **inaccuracy** of tracing result

- **Principles of reducing tracing overhead**
  - Use Low-level signal if possible
  - Defer expensive computation to the end of each path
  - Avoid memory related operation
void f1() {
    ...
    f2();
    ...
    call signal
    execution
    return signal
}

EIP : 0x02
ret : 0x02
time : x

EIP : 0x03
ret : 0x01
time : 10

EIP : 0x02
ret : 0x01
time : 45

EIP : 0x03
ret : 0x02
time : x

Call List

Return List

Trace Latency + Construct Call Chain
Besides latency and call stack, we also trace:
  - The # of instructions, system calls, file I/O calls, I/O traffic and etc.
  - We call them logical cost metrics

Some specious configurations are not obvious in latency

Logical metrics can capture subtle effect and are independent to the environment
Violet Overview

System code

Violet Hook

Static analyzer

Violet Tracer

S2E

Trace Analyzer

Path Comparison

Config dependency file

Configuration Constraints | Cost | Workload
--- | --- | ---
auto \( l > 0 \) & flush = 1 | fit_flush | SQL \( \Rightarrow \) INSERT
auto \( l > 0 \) & flush = 2 | NULL | SQL \( \Rightarrow \) INSERT
auto \( l > 0 \) & flush \( l = 2 \) | NULL | SQL \( \Rightarrow \) INSERT
auto \( l = 0 \) | NULL | SQL \( \Leftrightarrow \) INSERT
Generate Performance Impact Model

```
int write_row() {
    if (autocommit) {
        ...
        trx_commit_complete();
    } else {
        trx_mark_sql_stat_end();
    }
}

ulint trx_commit_complete() {
    if (flush_at_trx_commit==1) {
        log_group_write_buf();
        fil_flush();
    } else if (flush_at_trx_commit==2) {
        log_group_write_buf();
    } else {
        /* do nothing */
    }
}
```

<table>
<thead>
<tr>
<th>Constraints</th>
<th>Cost</th>
<th>Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto == 0</td>
<td>X</td>
<td>SQL == ALL</td>
</tr>
<tr>
<td>auto != 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Configuration
- Costly operation

```
Generate Performance Impact Model

```c
int write_row() {
    if (autocommit) {
        ...
        trx_commit_complete();
    } else {
        trx_mark_sql_stat_end();
    }
}

ulint trx_commit_complete() {
    if (flush_at_trx_commit==1) {
        log_group_write_buf();
        fil_flush();
    } else if (flush_at_trx_commit==2) {
        log_group_write_buf();
    } else {
        /* do nothing */
    }
}
```

### Constraints

<table>
<thead>
<tr>
<th>Constraints</th>
<th>Cost</th>
<th>Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto == 0</td>
<td>X</td>
<td>SQL == ALL</td>
</tr>
<tr>
<td>auto != 0 &amp; flush == 1</td>
<td>fil_flush+X</td>
<td>SQL == INSERT</td>
</tr>
</tbody>
</table>
Generate Performance Impact Model

```c
int write_row() {
    if (autocommit) {
        ...
        trx_commit_complete();
    } else {
        trx_mark_sql_stat_end();
    }
}

ulint trx_commit_complete() {
    if (flush_at_trx_commit==1) {
        log_group_write_buf();
        fil_flush();
    } else if (flush_at_trx_commit==2) {
        log_group_write_buf();
    } else {
        /* do nothing */
    }
}
```

<table>
<thead>
<tr>
<th>Constraints</th>
<th>Cost</th>
<th>Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto == 0</td>
<td>X</td>
<td>SQL == ALL</td>
</tr>
<tr>
<td>auto != 0 &amp; flush == 1</td>
<td>fil_flush+X</td>
<td>SQL == INSERT</td>
</tr>
</tbody>
</table>
Generate Performance Impact Model

<table>
<thead>
<tr>
<th>Constraints</th>
<th>Cost</th>
<th>Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto == 0</td>
<td>X</td>
<td>SQL == ALL</td>
</tr>
<tr>
<td>auto != 0 &amp; flush == 1</td>
<td>fil_flush+X</td>
<td>SQL == INSERT</td>
</tr>
<tr>
<td>auto != 0 &amp; flush == 2</td>
<td>X</td>
<td>SQL == INSERT</td>
</tr>
<tr>
<td>auto != 0 &amp; flush != 2</td>
<td>X</td>
<td>SQL == INSERT</td>
</tr>
</tbody>
</table>
Performance Comparison

• Compare the cost between each pair

<table>
<thead>
<tr>
<th>Constraints</th>
<th>Cost</th>
<th>Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto == 0 &amp; flush==1</td>
<td>X</td>
<td>SQL == ALL</td>
</tr>
<tr>
<td>auto == 0 &amp; flush==2</td>
<td>X</td>
<td>SQL == ALL</td>
</tr>
<tr>
<td>auto==0 &amp; flush!=2</td>
<td>X</td>
<td>SQL == ALL</td>
</tr>
<tr>
<td>auto!=0 &amp; flush==1</td>
<td>fil_flush+X</td>
<td>SQL == INSERT</td>
</tr>
<tr>
<td>auto!=0 &amp; flush==2</td>
<td>X</td>
<td>SQL == INSERT</td>
</tr>
<tr>
<td>auto!=0 &amp; flush!=2</td>
<td>X</td>
<td>SQL == INSERT</td>
</tr>
</tbody>
</table>
### Performance Comparison

- **Compare the cost between each pair**

<table>
<thead>
<tr>
<th>Constraints</th>
<th>Cost</th>
<th>Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto == 0 &amp; flush==1</td>
<td>X</td>
<td>SQL == ALL</td>
</tr>
<tr>
<td>auto == 0 &amp; flush==2</td>
<td>X</td>
<td>SQL == ALL</td>
</tr>
<tr>
<td>auto==0 &amp; flush!=2</td>
<td>X</td>
<td>SQL == ALL</td>
</tr>
<tr>
<td>auto!=0 &amp; flush==1</td>
<td>fil_flush+X</td>
<td>SQL == INSERT</td>
</tr>
<tr>
<td>auto!=0 &amp; flush==2</td>
<td>X</td>
<td>SQL == INSERT</td>
</tr>
<tr>
<td>auto!=0 &amp; flush!=2</td>
<td>X</td>
<td>SQL == INSERT</td>
</tr>
</tbody>
</table>
Performance Comparison

- Some path comparisons are not very meaningful

<table>
<thead>
<tr>
<th>Constraints</th>
<th>Cost</th>
<th>Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto == 0 &amp; flush==1</td>
<td>X</td>
<td>SQL == ALL</td>
</tr>
<tr>
<td>auto == 0 &amp; flush==2</td>
<td>X</td>
<td>SQL == ALL</td>
</tr>
<tr>
<td>auto == 0 &amp; flush!=2</td>
<td>X</td>
<td>SQL == ALL</td>
</tr>
<tr>
<td>auto!=0 &amp; flush==1</td>
<td>fil_flush+X</td>
<td>SQL == INSERT</td>
</tr>
<tr>
<td>auto!=0 &amp; flush==2</td>
<td>X</td>
<td>SQL == INSERT</td>
</tr>
<tr>
<td>auto!=0 &amp; flush!=2</td>
<td>X</td>
<td>SQL == INSERT</td>
</tr>
</tbody>
</table>

path 1: auto == 0 & flush == 2
path 2: auto != 0 & flush == 1
“Similar” Path First Comparison

• The paths with the most “similar” constraint compare first
  o If a constrain appears in both state, add one to similarity score

• If two paths don’t have common constraint
  o Don’t compare them
Implementation

• **Violet components are mostly written in C/C++**
  o Violet tracer is implemented as S2E plugins
  o Violet static analyzer is built on top of LLVM

• **S2E [ASPLOS ’11]**
  o Symbolic execution platform
  o Fast, in-vivo
Selective Symbolic Execution

• Complex constraint and path explosion

• Selective symbolic execution
  o Silently concretize variable before library call or syscall
  o Accurate but not complete
  o Relax rules to achieve good completeness
Outline

- Motivation
- Specious Configuration Code Patterns
- Violet Overview
- Evaluation
Evaluation Questions

❖ How effective is Violet in detecting specious configurations and unknow cases.

❖ How useful is Violet?

❖ What is the performance of Violet?
Experiment Setup

- **Evaluated systems**
  - MySQL, PostgreSQL, Apache, Squid

- **The manual effort to add hook is small**

<table>
<thead>
<tr>
<th>Software</th>
<th>SLOC</th>
<th># of config</th>
<th>Line of Hook</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>1.2M</td>
<td>330</td>
<td>197</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>843K</td>
<td>294</td>
<td>165</td>
</tr>
<tr>
<td>Apache</td>
<td>199K</td>
<td>172</td>
<td>158</td>
</tr>
<tr>
<td>Squid</td>
<td>178K</td>
<td>327</td>
<td>96</td>
</tr>
</tbody>
</table>
## 17 Specious Configurations

<table>
<thead>
<tr>
<th>Application</th>
<th>Configuration Name</th>
<th>Data Type</th>
<th>Detect</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>autocommit</td>
<td>Boolean</td>
<td>✓</td>
</tr>
<tr>
<td>MySQL</td>
<td>query_cache_wlock_invalidate</td>
<td>Boolean</td>
<td>✓</td>
</tr>
<tr>
<td>MySQL</td>
<td>general_log</td>
<td>Boolean</td>
<td>✓</td>
</tr>
<tr>
<td>MySQL</td>
<td>query_cache_type</td>
<td>Enumeration</td>
<td>✓</td>
</tr>
<tr>
<td>MySQL</td>
<td>sync_binlog</td>
<td>Integer</td>
<td>✓</td>
</tr>
<tr>
<td>MySQL</td>
<td>innodb_log_buffer_size</td>
<td>Integer</td>
<td>✓</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>wal_sync_method</td>
<td>Enumeration</td>
<td>✓</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>archive_mode</td>
<td>Enumeration</td>
<td>✓</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>max_wal_size</td>
<td>Integer</td>
<td>✓</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>checkpoint_completion_target</td>
<td>Float</td>
<td>✓</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>bgwriter_lru_multiplier</td>
<td>Float</td>
<td>✓</td>
</tr>
<tr>
<td>Apache</td>
<td>HostNamelookup</td>
<td>Enumeration</td>
<td>✓</td>
</tr>
<tr>
<td>Apache</td>
<td>Deny/Domain</td>
<td>Enum/String</td>
<td>✓</td>
</tr>
<tr>
<td>Apache</td>
<td>MaxKeepAliveRequests</td>
<td>Integer</td>
<td>×</td>
</tr>
<tr>
<td>Apache</td>
<td>KeepAliveTineOut</td>
<td>Integer</td>
<td>×</td>
</tr>
<tr>
<td>Squid</td>
<td>Cache</td>
<td>String</td>
<td>✓</td>
</tr>
<tr>
<td>Squid</td>
<td>Buffered_logs</td>
<td>Integer</td>
<td>✓</td>
</tr>
</tbody>
</table>
Discover New Specious Configuration

Specious configuration is 1) the setting whose default value causes performance regression; 2) some performance impact is not documented

<table>
<thead>
<tr>
<th>Application</th>
<th>Configuration Name</th>
<th>Performance Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>optimizer_search_depth</td>
<td>Default cause would cause bad performance for some join query</td>
</tr>
<tr>
<td>MySQL</td>
<td>concurrent_insert</td>
<td>Enable it would cause bad performance for read workload</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>vacuum_cost_delay</td>
<td>Default value is significantly worse than low values for write workload</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>archive_timeout</td>
<td>Small values cause performance penalties</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>random_page_cost</td>
<td>Value larger than 1.2 cause bad perf on SSD for join queries</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>log_statement</td>
<td>Setting mod cause bad perf for write workload when synchronous_commit is off</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>parallel_setup_cost</td>
<td>A higher value would avoid unnecessary parallelism</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>parallel_leader_participation</td>
<td>Enabling it can cause select join query to be slow</td>
</tr>
<tr>
<td>Squid</td>
<td>ipcache_size</td>
<td>The default value is relatively small and may cause performance reduction</td>
</tr>
<tr>
<td>Squid</td>
<td>cache_log</td>
<td>Enable cachelog with higher debug_option would cause extra I/O</td>
</tr>
<tr>
<td>Squid</td>
<td>store_objects_per_bucket</td>
<td>Decrease the setting would short the search time</td>
</tr>
</tbody>
</table>

8 new cases are confirmed by developers
Coverage Experiment for Violet

- MySQL: 51.2% (Analyzed) 71.4% (Total)
- PostgreSQL: 29.6% (Analyzed) 53.3% (Total)
- Apache: 29.6% (Analyzed)
- Squid:
  - Analyzed: 53.3%
  - Total: 53.9%

Total Configs: 53.9%
How Fast Is Violet

Analysis Time (seconds)

MySQL: 206
PostgreSQL: 117
Apache: 1171
Squid: 554
Related Work

• **Misconfiguration Detection**
  o Pcheck[OSDI’16], LearnConf[Eurosys’20], PeerPressure[OSDI’04], EnCore[ASPLOS’14]

• **Misconfiguration Diagnosis**
  o ConfAid[OSDI’10], X-ray[OSDI’12]

• **Performance Tuning**
  o Starfish [CIDR’11], Strider [LISA’03], SmartConf[ASPLOS’18]
Conclusion

1. Detecting specious configuration is a difficult task
2. Need to systematically reason about the performance effect of configuration from source code
3. Violet – an analytical approach to detect specious configuration in large system by symbolic execution
4. Detect 15 known specious configuration and 11 new cases

https://github.com/OrderLab/violet
Thank you!

Contact Information: hyigong1@jhu.edu