Effective Performance Issue Diagnosis with Value-Assisted Cost Profiling

Lingmei Weng, Columbia University
Yigong Hu, Johns Hopkins University
Peng Huang, University of Michigan
Jason Nieh, Columbia University
Junfeng Yang, Columbia University
A Real World Performance Issue in MariaDB

In v10.3.22, MariaDB crash recovery takes a long time

Server reads XXX LSN

Profilers are often recommended
Top ranked functions

```
3388 bool recv_group_scan_log_recs(lsn_t ckpt_lsn, ...) {
3417   ulint available_mem = srv_page_size *
3418       (buf_pool_get_n_pages() -
3419       (recv_n_pool_free_frames * srv_buf_pool_ins));
3424   do {
3431       recv_apply_hashed_log_recs(false);
3439       log.read_log_seg(&end_lsn, start_lsn + RSCAN_SIZE);
3440     } while (end_lsn != start_lsn &&
3441     !recv_scan_log_recs(available_mem, ...)
```

Problems

- Top ranked functions are not the culprit
- No info about buggy values that cause the issue

The diagnosis took 20+ days

function has more than 200 LOC  20+ branches

```
available_mem = 0
```
Key Insights

- **Function costs alone** are insufficient for performance diagnosis.

- **Dataflow is necessary** to understand root causes of performance issues:
  - a program variable’s values over time
  - useful to calibrate raw costs and identify problematic code
vProf Workflow

Static analysis on source code

1. Cost Profiling (GProf in libc.so)
2. Recording dataflow concurrently at profiling for selected variables

Selected variables

Debugging information in binary executable

Runtime locations

Runtime locations

Normal profile

Cost histogram

Value samples

Buggy profile

Cost calibration

Abnormal dataflow

Performance issue

Normal execution

Static analysis on source code

Selected variables

Debugging information in binary executable

Runtime locations

Cost histogram

Value samples

Buggy profile

Cost calibration

Abnormal dataflow

Normal profile

<table>
<thead>
<tr>
<th>function</th>
<th>adjusted cost</th>
<th>variable, bb, pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>recv_group_scan_log_recs</td>
<td>82.73</td>
<td>available_mem, bb3, WrongConstraint</td>
</tr>
</tbody>
</table>
vProf needs to address **three challenges**:

- **Pre-profiling**: select variables to minimize the overhead
- **Profiling**: record value samples concurrently and efficiently at profiling signal handler
- **Post-profiling**: effectively **leverage** recorded samples for diagnosis
Select Variables

✧ Focus value recording in a component related to the performance issue
  - e.g., storage/innobase/log

✧ Use static analysis to identify variables in code area that affects performance
  - conditional expression => operands
  - loop => induction variables
  - function call => parameters

```c
s = b + 3*i;
if (i < a.min) goto Lerr
while (i < a.length) {
    i = i + 2;
    s = s + 6;
   goo(ptr, s, i);
}
```
Access Selected Variables During Profiling

✦ Typical cost profiling is done by periodical sampling with signals
  - Profiling signals are delivered at different instruction addresses (PCs)

✦ Problems
  - accessible variables at different PCs changes
  - runtime locations for the same variable changes

✦ Solution
  - Fast index the runtime locations of accessible variables from arbitrary PCs
Hash Table was prepared in binary analysis step

Profiling signal at PC_2

while(a ...) {
    accessible variables
}

value of a, timestamp, PC_2

runtime locations of accessible variables at pc_2

DataFlow of Accessible Variables

Values ....
Discount the cost of inherent costly functions

- We calculate two kinds of discounts
  - (1) ranking discount; (2) variable discount
  - (2) is critical

Boost the cost of under-estimated functions

- little time but cause execution of other costly functions
Discount Calculation Needs A Baseline

- Discount for inherent costly functions
  - Compare to normal execution to identify inherent costly
    - Buggy execution
      - foo - 50s
      - ...
      - goo - 45s
      - boo - 6s
      - ...
    - Normal execution
      - foo - 49s
      - ...
      - goo - 35s
      - boo - 2s
      - ...
  - Baseline needs a similar use case, not necessarily identical
Ranking Discount

- Same rankings in buggy execution and normal execution
Similarity on distributions of values for variable $\text{var}$ in function $\text{goo}$

- Normal execution
- Buggy Execution

![Value vs Time Graph]

![Number of Value Graph]

**Similar value distributions of var**  \(\Rightarrow\)  

\[\text{adjusted\_cost}(\text{goo}) = (1 - \text{discount}) \times \text{profiling\_cost}(\text{goo})\]
Boost Under-Estimated Function Cost

♦ Samples outside current program, eg. dynamic libraries, are omitted.

♦ Values of the variables accessible from callers are also missed

✓ Virtually backtrace the call stack

• Costly library function;
• Abnormal value samples

Virtual profiling PC

Calls

Profiling signal PC
dyn libs

Buggy execution

...

goo - 9s
boo - 6s
foo - 1s
vProf Result for MariaDB Example

Function cost ranking in GProf

```
[1] recv_apply_hashed_log_recs
...
[445] recv_group_scan_log_recs
```

Calibrated function cost ranking in vProf

```
[1] recv_group_scan_log_recs
...
[3] recv_apply_hashed_log_recs
```

---

Discount

```
3388 bool recv_group_scan_log_recs(lsn_t ckpt_lsn, ...){
3417  ulint available_mem = srv_page_size *
3418   (buf_pool_get_n_pages() -
3419    (recv_n_pool_free_frames * srv_buf_pool_ins));
3424  do {
3431    recv_apply_hashed_log_recs(false);
3439    log.read_log_seq(&end_lsn, start_lsn + RSCAN_SIZE);
3440  } while (end_lsn != start_lsn &&
3441             !recv_scan_log_recs(available_mem, ...)
```

Boost
Besides cost calibration, vProf leverages the recorded value samples to provide further debugging aid:

✓ Identify the abnormal variables for a function

✓ Locate the code regions where abnormal values are accessed

✓ Infer potential performance bug patterns
Abnormal Value for MariaDB Example

Function cost ranking in vProf

[1]. recv_group_scan_log_recs
...

vProf debugging report

adjusted_cost: 87.73

suspicious_variable: available_mem
(abnormal_value: 0, location: bb3)...

bug_pattern: WrongConstraint

```c
3388 bool recv_group_scan_log_recs( lsn_t ckpt_lsn, ... ) {
3417   ulint available_mem = srv_page_size *
3418     (buf_pool_get_n_pages() -
3419      (recv_n_pool_free_frames * srv_buf_pool_ins));
3424   do {  
3431     recv_apply_hashed_log_recs(false);
3439     log.read_log_seg(&end_lsn, start_lsn + RSCAN_SIZE);
3440   } while (end_lsn != start_lsn &&
3441     !recv_scan_log_recs(available_mem, ...)
```
Evaluate vProf

✦ How effective vProf is?
✦ What is the advantages of vProf compared to other tools?
✦ Is vProf efficient enough to be practical?

Evaluation Settings:
• Intel Core i5 and 48GB DRAM
• Apply vProf to real-world performance bugs via LD_PRELOAD
• No instrumentation to applications
## Real-World Performance Issues

### All ground truth has already known in their bug reports

<table>
<thead>
<tr>
<th>ID</th>
<th>Apps</th>
<th>Bug Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>MariaDB</td>
<td>Server crash recovery loops on the same log sequence number</td>
</tr>
<tr>
<td>B2</td>
<td>MariaDB</td>
<td>Performance drops when the size of dataset is larger than the size of buffer pool</td>
</tr>
<tr>
<td>B3</td>
<td>MariaDB</td>
<td>Deleting a table with CASCADE constraint is very slow</td>
</tr>
<tr>
<td>B4</td>
<td>MariaDB</td>
<td>Slow start-up even when .ibd file validation is off</td>
</tr>
<tr>
<td>B5</td>
<td>MariaDB</td>
<td>Checking the server status takes &gt;10 seconds with 3M tables</td>
</tr>
<tr>
<td>B6</td>
<td>Apache httpd</td>
<td>Output filter endless loop so server process never terminates</td>
</tr>
<tr>
<td>B7</td>
<td>Apache httpd</td>
<td>Gracefully restart service with mmm-workers takes long time</td>
</tr>
<tr>
<td>B8</td>
<td>Apache httpd</td>
<td>Health check is executed more often than configured intervals</td>
</tr>
<tr>
<td>B9</td>
<td>Apache httpd</td>
<td>Slow startup/reload when many ghosts are configured</td>
</tr>
<tr>
<td>B10</td>
<td>Apache httpd</td>
<td>Workers take 60-100% CPU even though no client sent requests</td>
</tr>
<tr>
<td>B11</td>
<td>Redis</td>
<td>Cluster nodes command is costly in a large cluster</td>
</tr>
<tr>
<td>B12</td>
<td>Redis</td>
<td>BRPOP command becomes slow when a large number of clients</td>
</tr>
<tr>
<td>B13</td>
<td>Redis</td>
<td>ZREVRANGE command is 50% slower after upgrade</td>
</tr>
<tr>
<td>B14</td>
<td>PostgresSQL</td>
<td>EXPLAIN hangs for generating some query plans</td>
</tr>
<tr>
<td>B15</td>
<td>PostgresSQL</td>
<td>Vacuum process fails to prune all heap pages and endlessly retries</td>
</tr>
</tbody>
</table>
Effectiveness

<table>
<thead>
<tr>
<th>ID</th>
<th>vProf</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>1</td>
</tr>
<tr>
<td>B2</td>
<td>1</td>
</tr>
<tr>
<td>B3</td>
<td>1</td>
</tr>
<tr>
<td>B4</td>
<td>3</td>
</tr>
<tr>
<td>B5</td>
<td>4</td>
</tr>
<tr>
<td>B6</td>
<td>5</td>
</tr>
<tr>
<td>B7</td>
<td>3</td>
</tr>
<tr>
<td>B8</td>
<td>1</td>
</tr>
<tr>
<td>B9</td>
<td>2</td>
</tr>
<tr>
<td>B10</td>
<td>1</td>
</tr>
<tr>
<td>B11</td>
<td>1</td>
</tr>
<tr>
<td>B12</td>
<td>1</td>
</tr>
<tr>
<td>B13</td>
<td>2</td>
</tr>
<tr>
<td>B14</td>
<td>4</td>
</tr>
<tr>
<td>B15</td>
<td>3</td>
</tr>
</tbody>
</table>

✦ vProf ranks root causes of all 15 issues within the top 5
✦ 7 of 15 have their root causes ranked at the top 1
## Comparison with Other Tools

<table>
<thead>
<tr>
<th>ID</th>
<th>vProf</th>
<th>gprof</th>
<th>Perf</th>
<th>Perf-pt</th>
<th>Coz</th>
<th>Statistical debugging</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>1</td>
<td>454</td>
<td>32</td>
<td>32</td>
<td>NR</td>
<td>4</td>
</tr>
<tr>
<td>B2</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>NR</td>
<td>12</td>
</tr>
<tr>
<td>B3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>B4</td>
<td>3</td>
<td>21</td>
<td>9</td>
<td>5</td>
<td>NR</td>
<td>18</td>
</tr>
<tr>
<td>B5</td>
<td>4</td>
<td>13</td>
<td>4</td>
<td>9</td>
<td>NR</td>
<td>566</td>
</tr>
<tr>
<td>B6</td>
<td>5</td>
<td>36</td>
<td>13</td>
<td>13</td>
<td>NR</td>
<td>NR</td>
</tr>
<tr>
<td>B7</td>
<td>3</td>
<td>182</td>
<td>1024</td>
<td>1024</td>
<td>Crash</td>
<td>7</td>
</tr>
<tr>
<td>B8</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>ChildProc</td>
<td>3</td>
</tr>
<tr>
<td>B9</td>
<td>2</td>
<td>11</td>
<td>28</td>
<td>28</td>
<td>NR</td>
<td>9</td>
</tr>
<tr>
<td>B10</td>
<td>1</td>
<td>4</td>
<td>16</td>
<td>16</td>
<td>ChildProc</td>
<td>161</td>
</tr>
<tr>
<td>B11</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>10</td>
<td>2</td>
<td>NR</td>
</tr>
<tr>
<td>B12</td>
<td>1</td>
<td>5</td>
<td>19</td>
<td>19</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>B13</td>
<td>2</td>
<td>16</td>
<td>13</td>
<td>13</td>
<td>9</td>
<td>NR</td>
</tr>
<tr>
<td>B14</td>
<td>4</td>
<td>NR</td>
<td>163</td>
<td>163</td>
<td>ChildProc</td>
<td>13</td>
</tr>
<tr>
<td>B15</td>
<td>3</td>
<td>14</td>
<td>56</td>
<td>56</td>
<td>ChildProc</td>
<td>18</td>
</tr>
</tbody>
</table>


✦ Other tools rank root causes within the top 5 for at most 6 cases
### vProf is Effective in Diagnosing *Unresolved* Issues

<table>
<thead>
<tr>
<th>ID</th>
<th>Bug Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redis-10981</td>
<td><em>lrange</em> command takes longer to finish when Redis is upgrade from version 6.2.7 to 7.0.3</td>
<td>07-14-2022</td>
</tr>
<tr>
<td>MDEV-16289</td>
<td>Query runs unexpectedly slow. The query <em>selects</em> records created within a given time period and <em>excludes</em> the records that are referenced by another table in a another given period</td>
<td>05-25-2018</td>
</tr>
<tr>
<td>MDEV-17878</td>
<td>Searching for the query execution plan for a SELECT query involving many joins takes forever for larger datasets, using 100% CPU</td>
<td>11-30-2018</td>
</tr>
</tbody>
</table>

- All the above issues have both reporter and developer involving the debugging.
vProf is Efficient: CPU Overhead

- The overhead gaps between gprof and vProf are mostly within 5%
vProf is Efficient: Memory Overhead

<table>
<thead>
<tr>
<th>ID</th>
<th>#Vars</th>
<th>PCToVar(kB)</th>
<th>VariableArray(kB)</th>
<th>ValueSamples(kB)</th>
<th>Sum(kB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>233</td>
<td>3862</td>
<td>430</td>
<td>21133</td>
<td>25425</td>
</tr>
<tr>
<td>B2</td>
<td>65</td>
<td>4143</td>
<td>29</td>
<td>153</td>
<td>4325</td>
</tr>
<tr>
<td>B3</td>
<td>399</td>
<td>4005</td>
<td>26</td>
<td>38563</td>
<td>42594</td>
</tr>
<tr>
<td>B4</td>
<td>852</td>
<td>3987</td>
<td>67</td>
<td>58</td>
<td>4112</td>
</tr>
<tr>
<td>B5</td>
<td>577</td>
<td>3575</td>
<td>22</td>
<td>8</td>
<td>3605</td>
</tr>
<tr>
<td>B6</td>
<td>501</td>
<td>673</td>
<td>287</td>
<td>2</td>
<td>962</td>
</tr>
<tr>
<td>B7</td>
<td>113</td>
<td>162</td>
<td>6</td>
<td>16</td>
<td>184</td>
</tr>
<tr>
<td>B8</td>
<td>169</td>
<td>260</td>
<td>127</td>
<td>43</td>
<td>430</td>
</tr>
<tr>
<td>B9</td>
<td>374</td>
<td>194</td>
<td>16</td>
<td>25</td>
<td>235</td>
</tr>
<tr>
<td>B10</td>
<td>164</td>
<td>642</td>
<td>186</td>
<td>13</td>
<td>841</td>
</tr>
<tr>
<td>B11</td>
<td>531</td>
<td>612</td>
<td>382</td>
<td>1216</td>
<td>2210</td>
</tr>
<tr>
<td>B12</td>
<td>623</td>
<td>591</td>
<td>44</td>
<td>1755</td>
<td>2390</td>
</tr>
<tr>
<td>B13</td>
<td>564</td>
<td>641</td>
<td>754</td>
<td>132</td>
<td>1527</td>
</tr>
<tr>
<td>B14</td>
<td>479</td>
<td>2037</td>
<td>1031</td>
<td>79</td>
<td>3147</td>
</tr>
<tr>
<td>B15</td>
<td>805</td>
<td>2297</td>
<td>927</td>
<td>3269</td>
<td>6493</td>
</tr>
</tbody>
</table>

Min: 184kB
Max: 42MB
Conclusions

✧ Missing dataflow in profiler makes performance diagnosis ineffective

✧ vProf integrates dataflow to re-rank functions and reveal root cause

✧ vProf successfully diagnosed all 15 resolved performance issues and three unresolved performance issues

✧ The overhead of value-assisted profiling is acceptable

https://github.com/wenglingmei/vprofAE