APOLLO

AUTOMATIC DETECTION AND DIAGNOSIS OF PERFORMANCE REGRESSIONS IN DATABASE SYSTEMS

Jinho Jung, Hong Hu, Joy Arulraj, Taesoo Kim, Woonhak Kang*
APOLLO

• Holistic toolchain for debugging DBMS

1. Automatically find SQL queries exhibiting performance regressions

2. Automatically diagnose the root cause of performance regressions
MOTIVATION: DBMS COMPLEXITY

![Graph showing code size growth over years for PostgreSQL and SQLite.]

- PostgreSQL:
  - 2000: 6.1 MB
  - 2010: 26.4 MB
  - Present: 47.7 MB

- SQLite:
  - 2000: 1.4 MB
  - 2010: 4.4 MB
  - Present: 8.7 MB

Increase: 7x
MOTIVATION: PERFORMANCE REGRESSIONS

CHALLENGING TO BUILD SYSTEM WITH **PREDICTABLE** PERFORMANCE

RE: Query became very slow after 9.6

Query: https://pastebin.com/9b953tT7

It was running under 3 seconds (it's our default timeout) and now it takes 3 minutes.

MySQL

Bug #87164 Queries running much slower in version 5.7 versus 5.6

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Reporter: Alok Pathak</td>
<td>Email Updates: Subscribe</td>
</tr>
<tr>
<td>Status: Won't fix</td>
<td>Impact on me: None</td>
</tr>
<tr>
<td>Category: MySQL Server: Optimizer</td>
<td>Severity: S3 (Non-critical)</td>
</tr>
<tr>
<td>Version: 5.7, 5.7.18, 5.7.19</td>
<td>OS: CentOS (6 &amp; 7)</td>
</tr>
<tr>
<td>Assigned to: CPU Architecture: Any</td>
<td></td>
</tr>
</tbody>
</table>


Description:
After upgrading to MySQL 5.7, some queries are running very slow, taking abnormally long time in statistics.

When I run this query in 5.6, it finishes in less than a second but on MySQL 5.7, it's taking approx 3 minutes.
MOTIVATION: PERFORMANCE REGRESSIONS

CHALLENGING TO BUILD SYSTEM WITH PREDICTABLE PERFORMANCE

- Scenario: User upgrades a DBMS installation
  - Query suddenly takes 10 times longer to execute
  - Due to unexpected interactions between different components
  - Refer to this behavior as a performance regression

- Performance regression can hurt user productivity
  - Can easily convert an interactive query to an overnight one
MOTIVATION: PERFORMANCE REGRESSIONS

```
SELECT R0.S_DIST_06
FROM PUBLIC.STOCK AS R0
WHERE (R0.S_W_ID < CAST(LEAST(0, 1) AS INT8))
```
MOTIVATION: DETECTING REGRESSIONS

1 HOW TO DISCOVER QUERIES EXHIBITING REGRESSIONS?

Query runs slower on latest version

SELECT NO FROM ORDER AS R0
WHERE EXISTS (SELECT CNT FROM SALES AS R1
WHERE EXISTS (SELECT ID FROM HISTORY AS R2
WHERE (R0.INFO IS NOT NULL)));
MOTIVATION: REPORTING REGRESSIONS

2 HOW TO SIMPLIFY QUERIES FOR REPORTING REGRESSION?

Query runs slower on latest version

```
SELECT NO FROM ORDER AS R0
WHERE EXISTS (SELECT CNT FROM SALES AS R1
WHERE EXISTS (SELECT ID FROM HISTORY AS R2
WHERE (R0.INFO IS NOT NULL));
```
MOTIVATION: DIAGNOSING REGRESSIONS

3 HOW TO DIAGNOSE THE ROOT CAUSE OF THE REGRESSION?

Query runs slower on latest version

```
SELECT NO FROM ORDER AS R0
WHERE EXISTS (SELECT CNT FROM SALES AS R1
WHERE EXISTS (SELECT ID FROM HISTORY AS R2
WHERE (R0.INFO IS NOT NULL)));
```
1. **How to Discover Queries Exhibiting Regressions?**

**SQLFUZZ: Feedback-Driven Fuzzing**

**APOLLO TOOLCHAIN**

- Old Version
- SQLFUZZ
- SQLMIN
- SQLDEBUG
- New Version

**Bug Reports**
- Query
- Commit
- File list
- Function
APOLLO TOOLCHAIN

2 HOW TO SIMPLIFY QUERIES FOR REPORTING REGRESSION?

SQLMIN: BI-DIRECTIONAL QUERY REDUCTION ALGORITHMS

HOW TO SIMPLIFY QUERIES FOR REPORTING REGRESSION?

SQLFUZZ: BI-DIRECTIONAL QUERY REDUCTION ALGORITHMS

OLD VERSION

NEW VERSION

APOLLO TOOLCHAIN

SQLMIN

SQLDEBUG

BUG REPORTS

- Query
- Commit
- File list
- Function

JINHO JUNG (JINHO.JUNG@GATECH.EDU)
HOW TO DIAGNOSE THE ROOT CAUSE OF THE REGRESSION?

SQLDEBUG: STATISTICAL DEBUGGING + COMMIT BISECTION

APOLLO TOOLCHAIN

OLD VERSION

NEW VERSION

SQLFUZZ → SQLMIN → SQLDEBUG

BUG REPORTS
- Query
- Commit
- File list
- Function
TALK OVERVIEW

APOLLO TOOLCHAIN

OLD VERSION

NEW VERSION

SQLFUZZ

SQLMIN

SQLDEBUG

BUG REPORTS

- Query
- Commit
- File list
- Function

JINHO JUNG (JINHO.JUNG@GATECH.EDU)
#1: SQLFUZZ — DETECTING REGRESSIONS

SQLFuzz

1. Query Generator
   - Random queries

2. Query Executor
   - Candidate queries

3. Bug Validator
   - Queries exhibiting performance regression

Update SQL grammar probability table

OLD VERSION

NEW VERSION

JINHO JUNG (JINHO.JUNG@GATECH.EDU)
#1: SQLFUZZ — DETECTING REGRESSIONS

1. **QUERY GENERATOR: RANDOM QUERY GENERATION**

Retrieve schema → **Query Generator** → Valid queries → Check complexity → Queries for fuzzing

<table>
<thead>
<tr>
<th>SQL Grammar</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>0.3</td>
</tr>
<tr>
<td>LEFT JOIN</td>
<td>0.3</td>
</tr>
<tr>
<td>LIMIT</td>
<td>0.2</td>
</tr>
<tr>
<td>CAST</td>
<td>0.2</td>
</tr>
</tbody>
</table>

**SQL grammar probability table**

JINHO JUNG (JINHO.JUNG@GATECH.EDU)
#1: SQLFUZZ — DETECTING REGRESSIONS

## QUERY EXECUTOR: FEEDBACK-DRIVEN FUZZING

### OLD VERSION

**Query Executor**

### NEW VERSION

**Found Regression?**

**Update table**

**SQL grammar probability table**

```sql
SELECT R0.S_DIST_06
FROM PUBLIC.STOCK AS R0
WHERE (R0.S_W_ID < CAST (LEAST(0, 1) AS INT8))
```

**CASE** | **LEFT JOIN**
---|---
**LIMIT** | **CAST** +0.1
#1: SQLFUZZ — DETECTING REGRESSIONS

## REGRESSION VALIDATOR: REDUCING FALSE POSITIVES

### Filtering rules

<table>
<thead>
<tr>
<th>#</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Non-deterministic behavior?</td>
</tr>
<tr>
<td>2</td>
<td>Non-executed plan?</td>
</tr>
<tr>
<td>3</td>
<td>Usage of catalog statistics?</td>
</tr>
<tr>
<td>4</td>
<td>Enough memory?</td>
</tr>
<tr>
<td>5</td>
<td>Limit statement?</td>
</tr>
<tr>
<td>6</td>
<td>Query is too complex?</td>
</tr>
<tr>
<td>7</td>
<td>...</td>
</tr>
</tbody>
</table>

Regression Query → Filtering rules → Report → Developers

JINHO JUNG (JINHO.JUNG@GATECH.EDU)
TALK OVERVIEW

OLD VERSION

NEW VERSION

APOLLO TOOLCHAIN

SQLFUZZ

SQLMIN

SQLDEBUG

BUG REPORTS
- Query
- Commit
- File list
- Function

JINHO JUNG (JINHO.JUNG@GATECH.EDU)
• Bottom-up Query Reduction
  ▫ Extract valid sub-query

• Top-down Query Reduction
  ▫ Iteratively removes unnecessary expressions
SELECT S1.C2
FROM (  
SELECT CASE WHEN EXISTS (  
SELECT S0.C0  
FROM ORDER AS R1  
WHERE ((S0.C0 = 10) AND (S0.C1 IS NULL))  
) THEN S0.C0 END AS C2,  
FROM (  
SELECT R0.I_PRICE AS C0, R0.I_DATA AS C1,  
(SELECT ID FROM ITEM) AS C2  
FROM ITEM AS R0  
WHERE R0.PRICE IS NOT NULL  
OR (R0.PRICE IS NOT S1.C2)  
LIMIT 1000) AS S0) AS S1;
SELECT S1.C2
FROM (SELECT
    CASE WHEN EXISTS (SELECT S0.C0
        FROM ORDER AS R1
        WHERE ((S0.C0 = 10) AND (S0.C1 IS NULL))
    ) THEN S0.C0 END AS C2,
FROM (SELECT R0.I_PRICE AS C0, R0.I_DATA AS C1,
    (SELECT ID FROM ITEM) AS C2
FROM ITEM AS R0
WHERE R0.PRICE IS NOT NULL
OR (R0.PRICE IS NOT S1.C2)
LIMIT 1000) AS S0) AS S1;
SELECT S1.C2
FROM (SELECT CASE WHEN EXISTS (SELECT S0.C0 FROM ORDER AS R1 WHERE ((S0.C0 = 10) AND (S0.C1 IS NULL))) THEN S0.C0 END AS C2,
FROM (SELECT R0.I_PRICE AS C0, R0.I_DATA AS C1, (SELECT ID FROM ITEM) AS C2
FROM ITEM AS R0
WHERE R0.PRICE IS NOT NULL
OR (R0.PRICE IS NOT S1.C2)
LIMIT 1000) AS S0) AS S1;
SELECT CASE WHEN EXISTS ( SELECT S0.C0 FROM ORDER AS R1 WHERE ((S0.C0 = 10)) ) THEN S0.C0 END AS C2,
FROM ( SELECT R0.I_PRICE AS C0, FROM ITEM AS R0 WHERE R0.PRICE IS NOT NULL) AS S0) AS S1;
TALK OVERVIEW

OLD VERSION

NEW VERSION

APOLLO TOOLCHAIN

SQLFUZZ

SQLMIN

SQLDEBUG

BUG REPORTS

- Query
- Commit
- File list
- Function

JINHO JUNG (JINHO.JUNG@GATECH.EDU)
#3: SQLDEBUG — DIAGNOSING REGRESSIONS

SQLDEBUG

Regression query

Partially Reduced queries

DBMS

Slow
Fast

Control-flow Graphs (Traces)

First commit exhibiting regression?

Statistical Debugger

Commit bisection

BUG REPORTS

- Query
- Commit
- File list
- Function

JINHO JUNG (JINHO.JUNG@GATECH.EDU)
1. COMMIT BISECTION: FIND EARLIEST PROBLEMATIC COMMIT

- COMMIT 1: OLD VERSION (FAST QUERY EXECUTION)
- COMMIT 2
- COMMIT 3
- COMMIT 5: NEW VERSION (SLOW QUERY EXECUTION)

PROBLEM BEGINS HERE!
#3: SQLDEBUG — DIAGNOSING REGRESSIONS

## QUERY REDUCTION: PARTIALLY REDUCED QUERIES

### Original query

```
SELECT NO FROM ORDER AS R0 WHERE EXISTS (SELECT CNT FROM SALES AS R1 WHERE EXISTS (SELECT ID FROM
```

### Partially reduced queries

```
SELECT NO FROM ORDER AS R0 WHERE EXISTS (SELECT CNT FROM SALES AS R1 WHERE EXISTS (SELECT ID FROM
```

### Minimized query

```
SELECT CNT FROM SALES WHERE CNT > ID
```

---

JINHO JUNG (JINHO.JUNG@GATECH.EDU)

---

Ready to use statistical debugging?
#3: SQLDEBUG — DIAGNOSING REGRESSIONS

3 CONTROL-FLOW GRAPH COMPARISON: ALIGN TRACES

Functions

OLD VERSION

int func()
{
    if (cond1)
        work;
}

NEW VERSION

int func()
{
    if (cond1)
        work;
}
#3: SQLDEBUG — DIAGNOSING REGRESSIONS

3 CONTROL-FLOW GRAPH COMPARISON: ALIGN TRACES

OLD VERSION

```c
int func()
{
    if (cond1)
    work;
}
```

Traces

0x400
0x420 ➔ TRUE

NEW VERSION

```c
int func()
{
    if (cond1)
    work;
}
```

Traces

0x500
0x520 ➔ FALSE
### #3: SQLDEBUG — DIAGNOSING REGRESSIONS

#### CONTROL-FLOW GRAPH COMPARISON: ALIGN TRACES

<table>
<thead>
<tr>
<th>OLD VERSION</th>
<th>NEW VERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Functions</strong></td>
<td><strong>Functions</strong></td>
</tr>
</tbody>
</table>
| `int func(){
if (cond1)
work;
}` | `int func(){
if (cond1)
work;
}` |
| **Traces** | **Traces** |
| `0x400
0x420 ➜ TRUE` | `0x500
0x520 ➜ FALSE` |
| **Trace Alignment** | **Trace Alignment** |
| `func + 0x0
func + 0x20 ➜ TRUE` | `func + 0x0
func + 0x20 ➜ FALSE` |
#3: SQLDEBUG — DIAGNOSING REGRESSIONS

4 STATISTICAL DEBUGGING: FAST AND SLOW QUERY TRACES

Fast query execution traces

<table>
<thead>
<tr>
<th>PRED.</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TAKEN</td>
</tr>
<tr>
<td>2</td>
<td>TAKEN</td>
</tr>
</tbody>
</table>

Statistical model

Slow query execution traces

<table>
<thead>
<tr>
<th>PRED.</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TAKEN</td>
</tr>
<tr>
<td>2</td>
<td>NOT TAKEN</td>
</tr>
</tbody>
</table>
#3: SQLDEBUG — DIAGNOSING REGRESSIONS

4 STATISTICAL DEBUGGING: FAST AND SLOW QUERY TRACES

Fast query execution traces

Slow query execution traces

<table>
<thead>
<tr>
<th>PRED.</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TAKEN</td>
</tr>
<tr>
<td>2</td>
<td>TAKEN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PRED.</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TAKEN</td>
</tr>
<tr>
<td>2</td>
<td>NOT TAKEN</td>
</tr>
</tbody>
</table>

Statistical model

<table>
<thead>
<tr>
<th>RANK</th>
<th>FILE</th>
<th>FUNCTION</th>
<th>LINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>foo.c</td>
<td>bar()</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Final report

JINHO JUNG (JINHO.JUNG@GATECH.EDU)
RECAP

APOLLO TOOLCHAIN

OLD VERSION

NEW VERSION

SQLFUZZ

SQLMIN

SQLDEBUG

BUG REPORTS

- Query
- Commit
- File list
- Function

JINHO JUNG (JINHO.JUNG@GATECH.EDU)
EVALUATION

• Tested database systems
  ▫ PostgreSQL, SQLite

• Binary instrumentation to get control flow graphs
  ▫ DynamoRIO instrumentation tool

• Evaluation
  ▫ Efficacy of SQLFuzz in detecting regressions?
  ▫ Efficacy of SQLMin in reducing queries?
  ▫ Accuracy of SQLDebug in diagnosing regressions?
#1: SQLFUZZ — DETECTING REGRESSIONS

Discovered 10 previously unknown, unique performance regressions.

Mean Performance Drop (Ratio)

Lower is Better

200x performance drop

<table>
<thead>
<tr>
<th></th>
<th>PostgreSQL</th>
<th>SQLite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance Drop (Ratio)</td>
<td>218</td>
<td>201</td>
</tr>
</tbody>
</table>
False Positives Queries (Percent)

Discovered Queries

SQLFuzz

Filtering rules remove almost all false positives

JINHO JUNG (JINHO.JUNG@GATECH.EDU)
#2: SQLMIN — REPORTING REGRESSIONS

<table>
<thead>
<tr>
<th>Query Size (Bytes)</th>
<th>Discovered Queries</th>
<th>SQLMin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower is Better</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1602</td>
<td>Significant reduction in query size</td>
<td></td>
</tr>
<tr>
<td>380</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

JINHO JUNG (JINHO.JUNG@GATECH.EDU)
Branch related to root cause
Correctly identified in all cases
(within top-3 ranked branches)

10 regressions

- FIRST RANKED BRANCH: 5
- SECOND RANKED BRANCH: 2
- THIRD RANKED BRANCH: 3

JINHO JUNG (JINHO.JUNG@GATECH.EDU)
CASE STUDY #1: OPTIMIZER UPDATE

SELECT COUNT (*)
FROM (SELECT R0.ID
     FROM CUSTOMER AS R0
     LEFT JOIN STOCK AS R1
     ON (R0.STREET = R1.DIST)
     WHERE R1.DIST IS NOT NULL) AS S0
WHERE EXISTS (SELECT ID FROM CUSTOMER);

• Due to a bug fix (for a correctness bug)
  ▫ Breaks query optimization
  ▫ Optimizer no longer transforms the LEFT JOIN operator

> 1000x slow down

LATEST VERSION OF SQLITE
SELECT R0.ID FROM ORDER AS R0
WHERE EXISTS (SELECT COUNT(*)
FROM (SELECT DISTINCT R0.ENTRY
FROM CUSTOMER AS R1
WHERE (FALSE)) AS S1);

• Hashed aggregation executor update
  ▫ Resulted in redundantly building hash tables

3x slow down
LATEST VERSION OF POSTGRESQL
CONCLUSION

• APOLLO (v1.0)
  ▫ Toolchain for detecting & diagnosing regressions
  ▫ Open-sourced: https://github.com/sslab-gatech/apollo

• Adding support for other types of bugs (v2.0)
  ▫ Correctness bugs
  ▫ Performance bugs
  ▫ Database corruption
CONCLUSION

• Interested in integrating APOLLO with more DBMSs
  ▫ Discovered > 5 performance regressions in CockroachDB
  ▫ Improve the toolchain based on developer feedback

• Automation will help reduce labor of developing DBMSs
  ▫ Developers get to focus on more important problems
Supported by:

Developers: