DEBUGGING YOUR DATABASE SYSTEM USING APOLLO

JOY ARULRAJ
GEORGIA TECH
APOLLO

• Holistic toolchain for debugging database systems
  ▫ Inspired by Jepsen

1 AUTOMATICALLY FIND SQL QUERIES EXHIBITING PERFORMANCE REGRESSIONS

2 AUTOMATICALLY DIAGNOSE THE ROOT CAUSE OF PERFORMANCE REGRESSIONS

JOY ARULRAJ (ARULRAJ@GATECH.EDU)
APOLLO (VLDB 2020)

APOLLO: Automatic Detection and Diagnosis of Performance Regressions in Database Systems

Jinho Jung  
Hong Hu  
Joy Arulraj  
Taesoo Kim  
Woonhak Kang

ABSTRACT

Traditional database management systems (DBMSs) have a proven record of delivering reliable and efficient database services, but it has been observed that the performance of DBMSs can degrade over time due to various reasons such as software updates, hardware changes, or user behavior. Performance regressions are a significant concern as they can lead to degraded user experience and increased costs for database administration. In this paper, we propose a system called APOLLO (Automatic Performance Regression Regression) that automatically detects and diagnoses performance regressions. APOLLO leverages data from DBMS' operational logs, system monitoring, and user behavior to identify performance anomalies and then uses data mining techniques and machine learning algorithms to predict and diagnose the root cause of these anomalies. The system also provides insights into the performance degradation trends and suggests mitigation strategies to database administrators. We evaluate APOLLO on a real-world database system and show that it can accurately detect and diagnose performance regressions with high precision and recall.

1. INTRODUCTION

Database management systems (DBMSs) are the critical component of modern data-intensive applications [9, 10, 11]. The performance of these systems is measured in terms of the time it takes to execute a query, the amount of data processed, and the resources consumed. Performance regressions are a significant concern as they can lead to degraded user experience and increased costs for database administration. In this paper, we propose a system called APOLLO (Automatic Performance Regression Regression) that automatically detects and diagnoses performance regressions. APOLLO leverages data from DBMS' operational logs, system monitoring, and user behavior to identify performance anomalies and then uses data mining techniques and machine learning algorithms to predict and diagnose the root cause of these anomalies. The system also provides insights into the performance degradation trends and suggests mitigation strategies to database administrators. We evaluate APOLLO on a real-world database system and show that it can accurately detect and diagnose performance regressions with high precision and recall.

This work is supported in part by the Australian Research Council under grants LP190100505 and DE150101530.

Further, the growing dependency on databases for critical applications has made it crucial to understand and mitigate performance regressions. APOLLO helps in understanding the root cause of these performance regressions and suggests mitigation strategies to database administrators. We evaluate APOLLO on a real-world database system and show that it can accurately detect and diagnose performance regressions with high precision and recall.
MOTIVATION: DBMS COMPLEXITY

![Graph showing code size trends over years.]

- **Code Size (MB):** Lower is Better
- **Release Year:**
  - 2000:
    - PostgreSQL: 6.1 MB
    - SQLite: 1.4 MB
  - 2010:
    - PostgreSQL: 26.4 MB
    - SQLite: 4.4 MB
  - Present:
    - PostgreSQL: 47.7 MB
    - SQLite: 8.7 MB

- **Comparison:** 7x increase

**MOTIVATION:**
- **PostgreSQL** and **SQLite**
- Joy Arulraj (arulraj@gatech.edu)
MOTIVATION: PERFORMANCE REGRESSIONS

• Challenging to build systems with predictable performance
  ▫ Due to complex interactions between different components

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

• Performance regressions can hurt user productivity
  ▫ Can easily covert an interactive query to an overnight one
SELECT R0.S_DIST_06
FROM PUBLIC.STOCK AS R0
WHERE (R0.S_W_ID < CAST(LEAST(0, 1) AS INT8))

>10,000x slowdown

MOTIVATION: PERFORMANCE REGRESSIONS

Due to a recent optimizer update
- New policy for choosing the scan algorithm
- Resulted in over-estimating the number of rows in the table
- Earlier version: Fast bitmap scan
- Latest version: Slow sequential scan
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 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: 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));
```

How to DIAGNOSE THE ROOT CAUSE OF THE regression?

Query runs slower on latest version

Joy Arulraj (arulraj@gatech.edu)
1 HOW TO DISCOVER QUERIES EXHIBITING REGRESSIONS?

SQLFUZZ: FEEDBACK-DRIVEN FUZZING

OLD VERSION

NEW VERSION

SQLFUZZ

SQLMIN

SQLDEBUG

BUG REPORTS
- QUERY
- COMMIT
- FILE
- FUNCTION

JOY ARULRAJ (ARULRAJ@GATECH.EDU)
How to simplify queries for reporting regressions?

SQLMIN: Bi-directional query reduction algorithms

Joy Arulraj (arulraj@gatech.edu)
HOW TO DIAGNOSE THE ROOT CAUSE OF THE REGRESSION?

SQLDEBUG: STATISTICAL DEBUGGING + COMMIT BISECTION

3

APOLLO TOOLCHAIN

OLD VERSION

NEW VERSION

SQLFUZZ

SQLMIN

SQLDEBUG

BUG REPORTS

- QUERY
- COMMIT
- FILE
- FUNCTION

JOY ARULRAJ (ARULRAJ@GATECH.EDU)
TALK OVERVIEW

APOLLO TOOLCHAIN

OLD VERSION

NEW VERSION

SQLFUZZ

SQLMIN

SQLDEBUG

BUG REPORTS

- QUERY
- COMMIT
- FILE
- FUNCTION

JOY ARULRAJ (ARULRAJ@GATECH.EDU)
#1: SQLFUZZ — DETECTING REGRESSIONS

SQLFUZZ

1. Query Generator
2. Query Executor
3. Reg. Validator

Old Version  New Version

Random Queries  Candidate Queries

Queries Exhibiting Performance Regression

Update SQL Grammar Probability Table

JOY ARULRAJ (ARULRAJ@GATECH.EDU)
#1: SQLFUZZ — DETECTING REGRESSIONS

## QUERY GENERATOR: RANDOM QUERY GENERATION

1. **Retrieve Schema**
2. **Query Generator**
3. **Check Complexity**
4. **Valid Queries**
5. **Queries for Fuzzing**

### SQL Grammar Probability Table

<table>
<thead>
<tr>
<th>CASE</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEFT JOIN</td>
<td>0.3</td>
</tr>
<tr>
<td>LIMIT</td>
<td>0.3</td>
</tr>
</tbody>
</table>

| LIMIT | 0.2 | CAST | 0.2 |

**CASE**: 0.3  **LEFT JOIN**: 0.3  **CAST**: 0.2
#1: SQLFUZZ — DETECTING REGRESSIONS

2 QUERY EXECUTOR: FEEDBACK-DRIVEN FUZZING

OLD VERSION

NEW VERSION

FOUND REGRESSION?

Query Executor

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

UPDATE TABLE

CASE | LEFT JOIN | CAST
LIMIT | +0.1

SQL Grammar Probability Table

JOY ARULRAJ (ARULRAJ@GATECH.EDU)
#1: SQLFUZZ — DETECTING REGRESSIONS

REGRESSION VALIDATOR: REDUCING FALSE POSITIVES

Filtering Rules

1. NON-DETERMINISTIC BEHAVIOR
2. NON-EXECUTED QUERY PLAN?
3. USAGE OF CATALOG STATISTICS?
4. ENOUGH MEMORY?
5. LIMIT STATEMENT?
6. QUERY IS TOO COMPLEX?
7. ...

Queries Exhibiting Performance Regression

DBMS Developers

Update Filtering Rules

JOY ARULRAJ (ARULRAJ@GATECH.EDU)
TALK OVERVIEW

APOLLO TOOLCHAIN

OLD VERSION

NEW VERSION

SQLFUZZ

SQLMIN

SQLDEBUG

BUG REPORTS

- QUERY
- COMMIT
- FILE
- FUNCTION

JOY ARULRAJ (ARULRAJ@GATECH.EDU)
#2: SQLMIN — REPORTING REGRESSIONS

• Top-Down Query Reduction
  ▫ Iteratively remove unnecessary query elements
• Bottom-Up Query Reduction
  ▫ Extract valid sub-queries
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;
#2: SQLMIN — REPORTING REGRESSIONS

```
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;
```

---

**TOP-DOWN REDUCTION REMOVE ELEMENTS**

- Remove conditions
- Remove columns
- Remove sub-queries
- Remove clauses
SELECT S1.C2
FROM (
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

APOLLO TOOLCHAIN

OLD VERSION

NEW VERSION

SQLFUZZ

SQLMIN

SQLDEBUG

BUG REPORTS

- QUERY
- COMMIT
- FILE
- FUNCTION

JOY ARULRAJ (ARULRAJ@GATECH.EDU)
#3: SQLDEBUG — DIAGNOSING REGRESSIONS

SQLDEBUG

Regression Query

SQLMin

Commit Bisection

DBMS

Slow

Fast

Partially Reduced Queries

Control-Flow Graphs (Traces)

Statistical Debugger

First Commit Exhibiting Regression?

BUG REPORT

- QUERY
- COMMIT
- FILE
- FUNCTION

JOY ARULRAJ (ARULRAJ@GATECH.EDU)
1. COMMIT BISECTION: FIND EARLIEST PROBLEMATIC COMMIT

- COMMIT 1
- COMMIT 2
- COMMIT 3
- COMMIT 5

OLD VERSION (FAST QUERY EXECUTION)

PROBLEM BEGINS HERE!

NEW VERSION (SLOW QUERY EXECUTION)

JOY ARULRAJ (ARULRAJ@GATECH.EDU)
#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 SALES WHERE CNT > ID))
```

**Partially Reduced Queries**
```
SELECT CNT FROM SALES WHERE EXISTS (SELECT ID FROM SALES WHERE CNT > ID)
```

**Minimized Query**
```
SELECT CNT FROM SALES WHERE CNT > ID
```
2 QUERY REDUCTION: PARTIALLY REDUCED QUERIES

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

COLLECT SET OF QUERIES

SELECT CNT FROM SALES WHERE CNT > ID
#3: SQLDEBUG — DIAGNOSING REGRESSIONS

## 3

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

![Diagram showing control-flow graph comparison]

**Functions**

Old Version

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

New Version

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

## CONTROL-FLOW GRAPH COMPARISON: ALIGN TRACES

### Functions

**Old Version**

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

**New Version**

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

### Traces

**Old Version**

- Traces: 0x400, 0x420
  - 0x420 → true

**New Version**

- Traces: 0x500, 0x520
  - 0x520 → false

JOY ARULRAJ (ARULRAJ@GATECH.EDU)
#3: SQLDEBUG — DIAGNOSING REGRESSIONS

## CONTROL-FLOW GRAPH COMPARISON: ALIGN TRACES

### Functions

**Old Version**

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

**New Version**

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

### Traces

**Old Version**

- `0x400`
- `0x420` → `true`

**New Version**

- `0x500`
- `0x520` → `false`

### Trace Alignment

**Old Version**

- `func + 0x0`
- `func + 0x20` → `true`

**New Version**

- `func + 0x0`
- `func + 0x20` → `false`

---

JOY ARULRAJ (ARULRAJ@GATECH.EDU)
#3: SQLDEBUG — DIAGNOSING REGRESSIONS

## 4

STATISTICAL DEBUGGING: FAST AND SLOW QUERY TRACES

<table>
<thead>
<tr>
<th>BRANCH</th>
<th>TRACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TAKEN</td>
</tr>
<tr>
<td>2</td>
<td>TAKEN</td>
</tr>
</tbody>
</table>

Fast Query Execution Traces

<table>
<thead>
<tr>
<th>BRANCH</th>
<th>TRACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TAKEN</td>
</tr>
<tr>
<td>2</td>
<td>NOT TAKEN</td>
</tr>
</tbody>
</table>

Slow Query Execution Traces

Statistical Debugging Model

JOY ARULRAJ (ARULRAJ@GATECH.EDU)
#3: SQLDEBUG — DIAGNOSING REGRESSIONS

4 STATISTICAL DEBUGGING: FAST AND SLOW QUERY TRACES

Fast Query Execution Traces

<table>
<thead>
<tr>
<th>BRANCH</th>
<th>TRACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TAKEN</td>
</tr>
<tr>
<td>2</td>
<td>TAKEN</td>
</tr>
</tbody>
</table>

Slow Query Execution Traces

<table>
<thead>
<tr>
<th>BRANCH</th>
<th>TRACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TAKEN</td>
</tr>
<tr>
<td>2</td>
<td>NOT TAKEN</td>
</tr>
</tbody>
</table>

Statistical Debugging Model

Bug Report

<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>
RECAP

APOLLO TOOLCHAIN

OLD VERSION

NEW VERSION

SQLFUZZ

SQLMIN

SQLDEBUG

BUG REPORTS

- QUERY
- COMMIT
- FILE
- FUNCTION

JOY ARULRAJ (ARULRAJ@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?
Discovered 10 previously unknown, unique performance regressions.
(7 acknowledged, 2 fixed)

Mean Performance Drop (Ratio)

Lower is Better

200x performance drop

PostgreSQL

SQLite

JOY ARULRAJ (ARULRAJ@GATECH.EDU)
#1: SQLFUZZ — FALSE POSITIVES

False Positive Queries (Percent)

Lower is Better

Filtering rules remove almost all false positives

<table>
<thead>
<tr>
<th>Discovered Queries</th>
<th>SQLFuzz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>0.0044</td>
</tr>
</tbody>
</table>

Joy Arulraj (arulraj@gatech.edu)
#2: SQLMIN — REPORTING REgressions

### Query Size (Bytes)
- **Lower is Better**

<table>
<thead>
<tr>
<th>Discovered Queries</th>
<th>SQLMin</th>
</tr>
</thead>
<tbody>
<tr>
<td>1602</td>
<td>380</td>
</tr>
</tbody>
</table>

**Significant reduction in query size**

77% reduction

Joy Arulraj (arulraj@gatech.edu)
#3: SQLDEBUG — DIAGNOSING REGRESSIONS

Branch related to root cause correctly identified in all cases (within top-3 ranked branches)

- First ranked branch: 5
- Second ranked branch: 2
- Third ranked branch: 3

10 regressions
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

• Regression status: Not Yet Fixed
  ▫ Searching for a fix that resolves both correctness and performance issues
CASE STUDY #2: EXECUTION ENGINE UPDATE

```
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
- Regression status: Fixed
  - If hash table already exists, then reuse the table

3x slowdown

LATEST VERSION OF POSTGRESQL

Joy Arulraj (arulraj@gatech.edu)
CONCLUSION

• APOLLO (v1.0)
  ▫ Toolchain for detecting & diagnosing regressions
  ▫ Going to be open-sourced in 2020

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

• Interested in integrating APOLLO with more database systems
  ▫ Improve the toolchain based on developer feedback
• Automation will help reduce labor cost of developing DBMSs
  ▫ Developers get to focus on more important problems
END

@joy_arulraj