Automatic Detection of Performance Bugs in Database Systems using Equivalent Queries

Xinyu Liu, Qi Zhou, Joy Arulraj, Alessandro Orso

The problem addressed in the paper

1. Developers extensively test DBMSs to improve reliability and accuracy
2. Such scrutiny has not been applied to performance bugs
3. Performance bugs affect the user experience

```
SELECT job, deptno FROM emp WHERE  job = 'Technical'
GROUP BY job, deptno LIMIT 13;
```

(a) Original query Q1, execution time = 13 s

```
SELECT CAST('Technical' AS VARCHAR(10)) AS "job", deptno
FROM emp WHERE "job" = 'Technical'
GROUP BY job, deptno LIMIT 13;
```

(b) Mutated query Q2, execution time = 9 ms

Motivation behind solving that problem

1. Using predetermined performance baseline to detect performance regressions is human-intensive and error-prone.
2. Using differential testing to detect the performance regressions
   - ●Require two versions of the DBMS
   - ●Focus on structurally simple queries specifically tailored for uncovering regressions

Proposed solution-AMOEBA

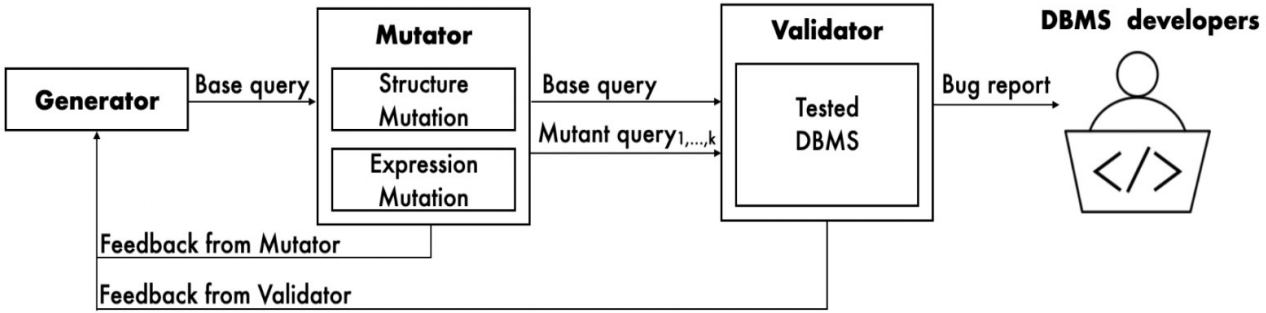


Figure 3: Architecture of AMOEBA– GENERATOR constructs a set of base SQL queries based on feedback from other components. MUTATOR performs semantic-preserving structural and expression mutations on the base queries. VALIDATOR executes a pair of semantically equivalent queries using the target DBMS and reports query pairs that exhibit a significant difference in runtime performance.

Proposed solution-AMOEBA

1. Construct a performance oracle by comparing the execution time of two semantically equivalent queries.
2. Construct queries tailored to discover the performance bugs
   - Incorporate the feedback mechanism
1. Introduce two types of semantic preserving mutation rules
   - structural mutation
   - expression mutation

Table 3: Illustrative List of Mutation Rules.

Category	Rule Idx	Transformation	Working Example
Structure	13	Push filter through GROUP BY	(t1.c GROUP BY t1.c) WHERE t1.c > 0 → (t1.c WHERE t1.c > 0) GROUP BY t1.C
	59	Propagate LIMIT through LEFT JOIN	(t1 LEFT JOIN t2) SORT t1.c → ((t1 SORT t1.c) LEFT JOIN t2) SORT t1.c
Expression	15	Convert EXTRACT into date range comparison	EXTRACT (YEAR FROM c1) < 2021 → c1 < TIMESTAMP '2021-01-01 00:00:00'
	54	Reduce expression in FILTER	WHERE c1 = CAST(10/2) as INT → WHERE c1 = 5

Evaluation in terms of the research questions addressed

RQ1. Can AMOEBA find performance bugs in DBMSs?

RQ2. How efficient is AMOEBA?

RQ3. Are all mutation rules created equal with respect to discover-ing performance bugs?

RQ4. How does AMOEBA compare against other techniques for finding performance bugs?

RQ5. How do the base queries in AMOEBA compare against those in Calcite?

Implementation

Query Generation:
The author implement the GENERATOR based on SQLALCHEMY.

Query Mutation: Build MUTATOR on top of the Calcite.

Implementation Scope: Four data types (integer, double, datetime, string), Several SQL constructs (Group, Distinct, Order, Union) and Functions (AVG, SUM)

Schema: SCOTT

RQ1. Can AMOEBA find performance bugs in DBMSs?

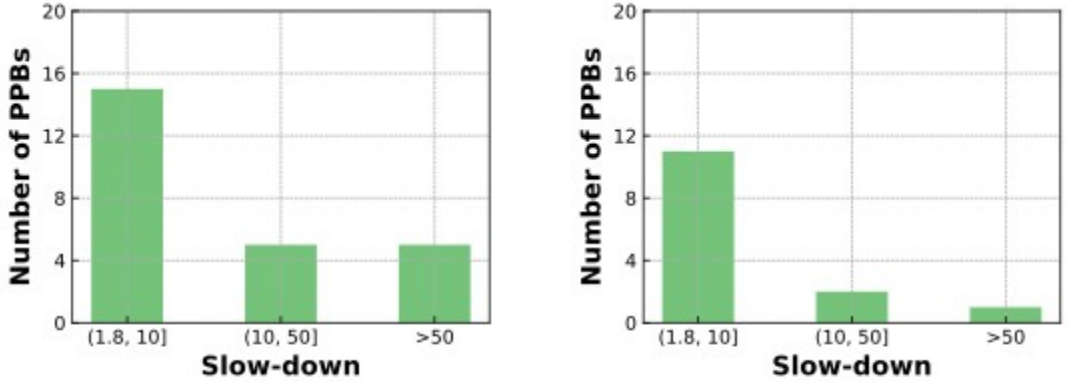


Figure 5: Runtime Impact of Potential Performance Bugs – Figures (a) and (b) show the slow-down of queries caused by PPBs for CockroachDB and PostgreSQL, respectively.

RQ1. Can AMOEBA find performance bugs in DBMSs?

Example 1.

```
/* [First query, 75 milliseconds] */
SELECT Max(emp.sal)
FROM  dept INNER JOIN emp ON ename NOT LIKE name
WHERE  name = 'ACCT';
/* [Second query, 238 milliseconds] */
SELECT Max(emp.sal)
FROM   dept INNER JOIN emp ON ename NOT LIKE name
WHERE  name = 'ACCT' IS TRUE;
```

RQ1. Can AMOEBA find performance bugs in DBMSs?

Example 2.

```
/* [First query, 7 milliseconds] */
SELECT sal FROM emp LEFT OUTER JOIN (SELECT job FROM
     bonus LIMIT 1) AS t ON true
WHERE t.job IS NOT DISTINCT FROM 'ACCT';
/* [Second query, 211 milliseconds] */
SELECT sal FROM emp WHERE (SELECT job FROM bonus LIMIT
     1) IS NOT DISTINCT FROM 'ACCT';
```

RQ1. Can AMOEBA find performance bugs in DBMSs?

Example 3.

```
/* [First query, 25 milliseconds] */
SELECT emp_pk FROM emp WHERE emp_pk > 100;
/* [Second query, 72 milliseconds] */
SELECT emp_pk FROM emp WHERE emp_pk > 100 GROUP BY
     emp_pk;
```

RQ2. How efficient is AMOEBA?

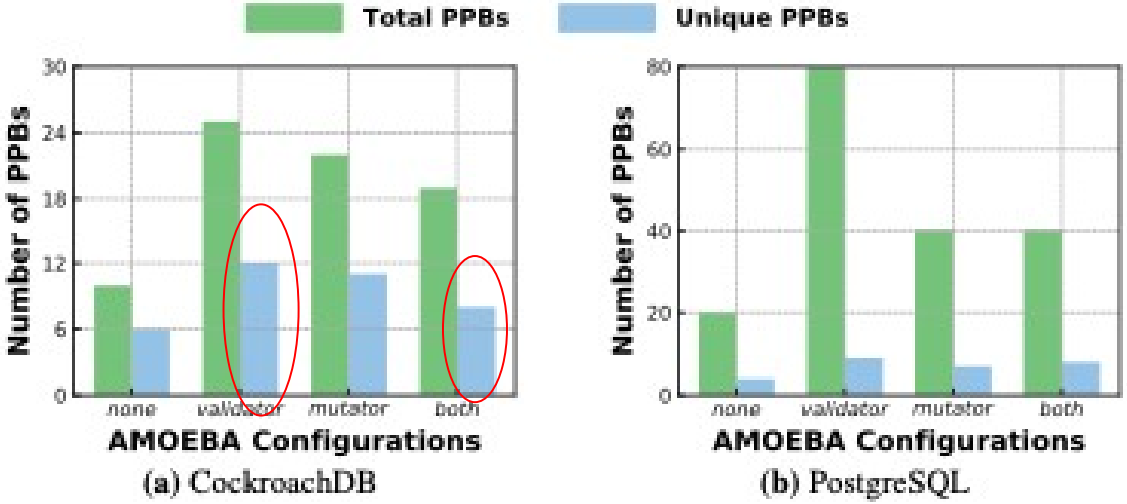


Figure 6: Efficiency of AMOEBA– Figures (a) and (b) show the number of total PPBs and unique PPBs that AMOEBA discovers after five hours in CockroachDB and PostgreSQL, respectively.

RQ3. Are all mutation rules created equal with respect to discovering performance bugs?

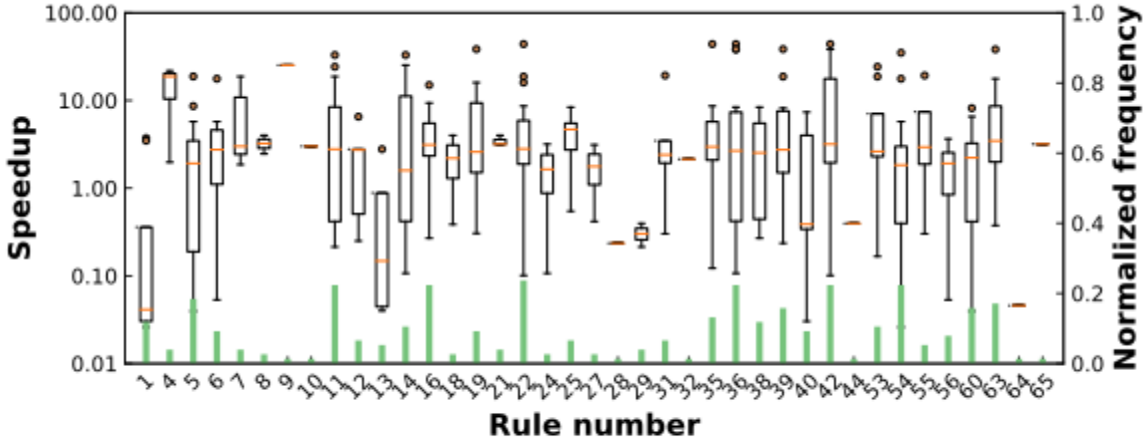


Figure 7: Impact of rules on query performance and frequency of bug-revealing query pairs.

RQ3. Are all mutation rules created equal with respect to discovering performance bugs?

Patterns result from compositional effects of mutation rules:
Contention between mutation rules
Enabling effect of mutation rules

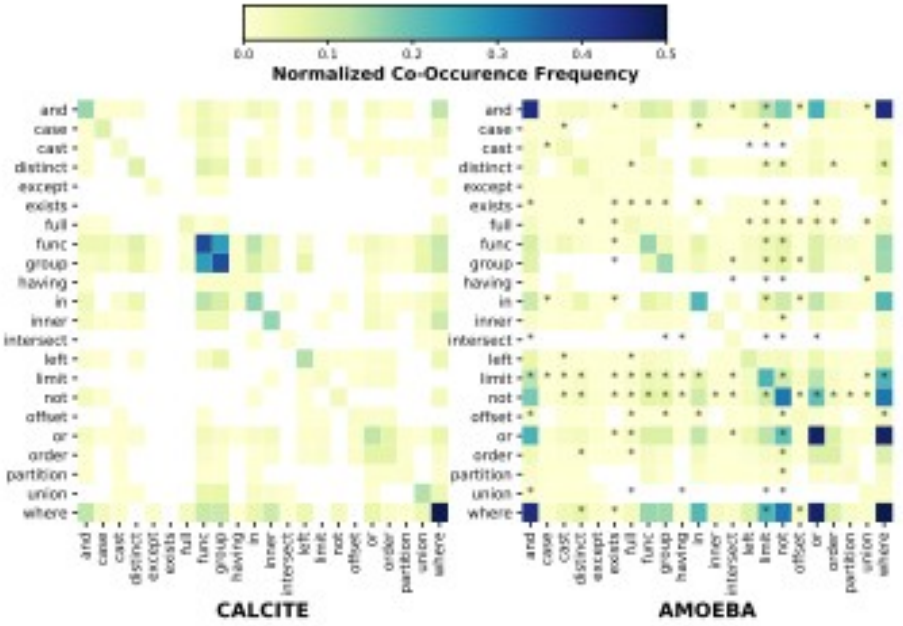
The characteristics of the mutation rules that generate query pairs that can reveal PPBs:
Mostly rearrange or eliminate expensive operators (UNION, GROUP BY, and JOIN)
Conversely, SELECT and ORDER BY are less likely to generate query pairs that trigger PPBs.

RQ4. How does AMOEBA compare against other techniques for finding performance bugs?

Table 4: Comparative Analysis of AMOEBA – Number of PPBs discovered by the set of query pairs in each baseline and by AMOEBA.

Benchmark	PPBs Found	
	Cockroach	PostgreSQL
Benchmark 1 (TLP)	1	1
Benchmark 2 (Calcite)	4	4
Benchmark 3 (Calcite+AMOEBA)	4	6
AMOEBA	25	14

RQ5. How do the base queries in AMOEBA compare against those in Calcite?



Novelty of the proposed solution considering state-of-the-art

State-of-the-art:
The use of standard benchmarks, predetermined performance baselines, comparison against previous DBMS versions, differential testing.

Novelty:
Semantically equivalent query pairs to detect performance discrepancies.
The feedback-driven query generation mechanism.

Assumptions made

- Semantically equivalent queries execute the same time on the same DBMS.
- Semantically equivalent queries, despite their distinct execution plans, are still capable of identifying PPBs.
- By using feedback-driven query mechanism, it can improve the possibility to find the PPBs: assume identified features will consistently cause performance problems across different DBMS implementations.

Limitations of the proposed solution

- AMOEBA relies on the Calcite framework, which has its limitations, but the extensibility of Calcite allows for future enhancements.
- For some reason, Calcite's rules fail to satisfy the property of preserving query semantics, then AMOEBA may generate false positives.
- The line between performance defects and optimization opportunities is narrow due to computational and design factors.
- Amoeba identifies relevant scenarios that developers should consider, but ultimately it is the developer's decision to support a given optimization.
- Automated query generation and changes can result in queries that appear artificial, and therefore may alienate developers rather than compel them to investigate and fix the corresponding issues.
- SQL allows developers to focus on logic rather than optimization, which means performance can vary greatly depending on the query formulation.

Practical significance of the proposed solution

- Improve DBMS performance by auto detecting hidden errors. Also reduce manual testing avoiding wasting more time and resources.
- Develop efficient applications by ensuring optimal performance of DBMS.
- Increase developer awareness of potential performance issues.
- Provide more valuable feedback and advices on DBMS design and optimization efforts.

Discussion points

- Detect the performance bugs between the DBMSs using the concept of Amoeba.
- Can we use LLM to generate the base queries?
- Can different indexing strategies affect the performance of amoebic approach usage in different systems?

Thank you for listening!