ClickHouse

Fuzzing databases is difficult

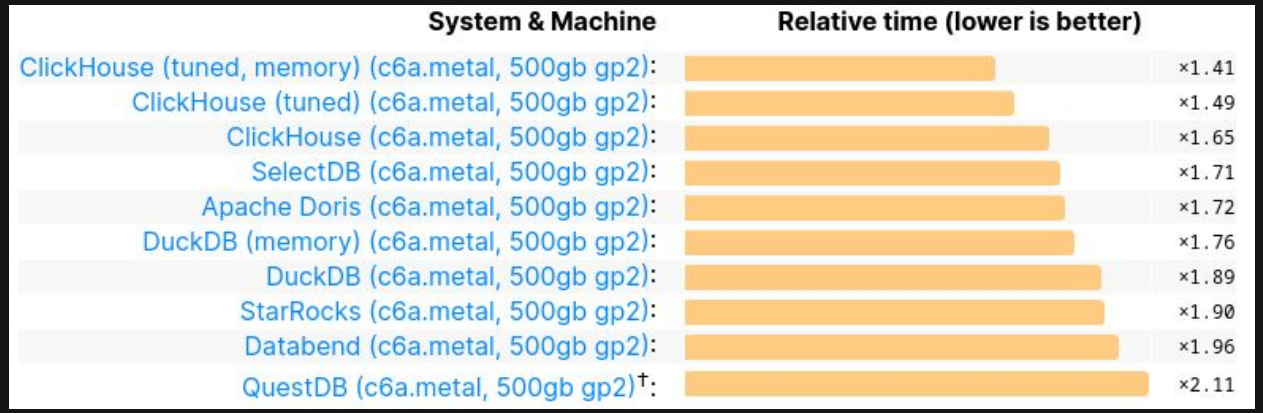
Pedro Ferreira - QA Engineer at ClickHouse

FOSDEM

Feb 1st 2025

What is ClickHouse?

- Open-source OLAP column store database written in C++.
- Can be self hosted or managed in ClickHouse Cloud.
- Very fast and resource efficient from one node to hundreds of nodes.



System & Machine	Relative time (lower is better)	
ClickHouse (tuned, memory) (c6a.metal, 500gb gp2):		×1.41
ClickHouse (tuned) (c6a.metal, 500gb gp2):		×1.49
ClickHouse (c6a.metal, 500gb gp2):		×1.65
SelectDB (c6a.metal, 500gb gp2):		×1.71
Apache Doris (c6a.metal, 500gb gp2):		×1.72
DuckDB (memory) (c6a.metal, 500gb gp2):		×1.76
DuckDB (c6a.metal, 500gb gp2):		×1.89
StarRocks (c6a.metal, 500gb gp2):		×1.90
Databend (c6a.metal, 500gb gp2):		×1.96
QuestDB (c6a.metal, 500gb gp2)†:		×2.11

Testing with Fuzzers

- Fuzzers are a hot research topic in many fields including databases.
- ClickHouse has embraced many Fuzzers over the years.
- The fuzzers include: SQLancer, AST Fuzzer, WINGFUZZ, AFL, libFuzzer and now BuzzHouse.
- All of them have their strengths and weaknesses.

Testing databases

- Databases do many operations: query and storage optimization, ACID properties, multiplex client connections.
- SQL language is extensive.
- State is kept between queries.
- Databases have to run for a long time without interruptions.
- Ensure correct results are returned.
- Evaluate performance.

What do we need?

- Attempt to write correct SQL statements by following grammar rules.
- Keep catalog information, in order to extend correctness.
- Update the catalog with changes on it.
- Don’t forget to provide random inputs sometimes. Too many rules make the fuzzer strict on what it can generate.
- Use oracles to find wrong results.

AST Fuzzer

- Developed by the ClickHouse team.
- Swaps internal Abstract Syntax Trees in the client before sending to the server.
- The input can be a test file, or user input from the terminal.
- https://clickhouse.com/blog/fuzzing-click-house

AST Fuzzer

```
CREATE TABLE test (c0 String, c1 Bool) ENGINE = MergeTree() ORDER BY tuple();
```

```
SELECT test.c0 FROM test WHERE c0 = 'a';
```

```
SELECT DISTINCT test.c0
FROM test
WHERE c0 = materialize('a')
```

```
EXPLAIN PIPELINE
SELECT DISTINCT test.c0
FROM test
WHERE c0 = 'a'
```

```
SELECT DISTINCT
    test.c0,
    test.c0
FROM test
ALL INNER JOIN test AS alias0 ON c0 = alias0.c0
WHERE c0 = 'a'
```

Designing a fuzzer

```
SELECT
      n_name, sum(l_extendedprice * (1 - l_discount)) AS revenue
FROM
      customer, orders, lineitem, supplier, nation, region
WHERE
      c_custkey = o_custkey AND l_orderkey = o_orderkey AND l_suppkey = s_suppkey
      AND c_nationkey = s_nationkey AND s_nationkey = n_nationkey AND n_regionkey = r_regionkey
      AND r_name = 'ASIA'
      AND o_orderdate >= DATE '1994-01-01' AND o_orderdate < DATE '1994-01-01' + INTERVAL '1' year
GROUP BY
      n_name
ORDER BY
      revenue DESC;
```

Let’s design a fuzzer

```
if 10% probability
  -   CREATE TABLE statement
else if 20% probability
  -   INSERT INTO a random table
else if 5% probability
  -   DELETE FROM a table
else if 10% probability
  -   ALTER TABLE statement
else if 5% probability
  -   DROP TABLE statement
else
  -   Run SELECT query
```

What are the issues?

- At every 20 statements, a DROP statement will be generated.
- The CREATE statement is generally more complex than DROP, so it’s more likely to fail and we may end without tables in the catalog!
- A table won’t last long in the catalog. In real life, a table stays forever!
- Some DELETES may clear tables completely, so we have to make sure tables contain data most of the time.

ClickHouse

So I came up with BuzzHouse

BuzzHouse by the numbers

24k

Lines of code

125+

Issues found

20+

Engineers busy

BuzzHouse in action

- Create a few tables during start, until the desired number of tables is reached.
- Generate queries based on successful catalog queries.
- Always keep a minimum number of tables in the catalog.
- Set a limit on the query size.
- Generate insert queries with values based on column types.
- Dump the session seed to reproduce it later.

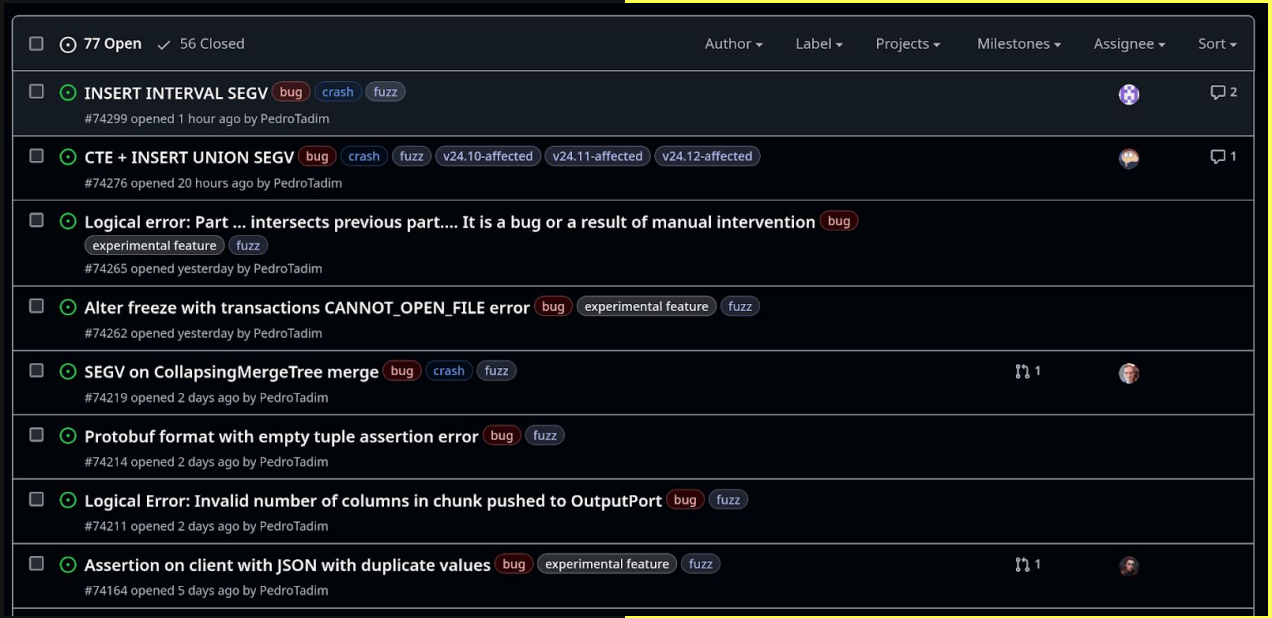
What BuzzHouse generates

```
SELECT
    f4(exists((
        SELECT CAST('-5705509103000988089', 'Int256')[4]
    )), t1d0.c1.`😆`, t1d0.c1.`😆`),
    toIntervalMonth(t1d0.c1.`😆`),
    t0d0.c1
FROM
(
    SELECT
        t0d1._part[4] AS c0,
        t0d1._part_data_version AS c1,
        rank() OVER () AS c2
    FROM t15 AS t0d1
    INNER JOIN d0.t4 AS t1d1 ON (t0d1._part = t1d1.c1) AND (t0d1.`c2.null` = t1d1.c1)
    WHERE t0d1._part_offset < randomString(780)
) AS t0d0
GLOBAL ANTI RIGHT JOIN t25 AS t1d0 ON t1d0.`c2.c4`.^`😆` = t0d0.c2
LEFT ARRAY JOIN t1d0.`c2.c5` AS c0
WHERE (t1d0.`c2.c4`, f0(c0, c0, t1d0.`c2.c5`), t1d0.`c2.c4`) NOT IN (
    SELECT NULL AS c0
    FROM t25 AS t0d1
    CROSS JOIN t15 AS t1d1
    CROSS JOIN t25 AS t2d1
    ALL FULL OUTER JOIN d0.t4 AS t3d1 ON t2d1.`c2.c5`
    ORDER BY ALL ASC
)
```


ClickHouse

BuzzHouse findings

- Crashes
- Logical errors
- Wrong results
- OOM kills
- Never ending queries



ClickHouse

Finding wrong results

- Dump a table and read it back again. Tests data formats.
- Run and compare equivalent queries using an oracle. This strategy was pioneered by SQLancer.
- Run the same query with different settings, such as the number of threads.
- Use a peer table from another database, and then compare computation results with the same ordering clause or global aggregate.

The combinations are too many!

- Many types including: Integers, Strings, Arrays, Tuples, Enums, the new JSON and Dynamic types.
- More than 1000 SQL functions and tunable settings.
- More than 30 table engines including MergeTree and its variations, Memory tables, S3 tables, tables from external databases such as MySQL.
- Tables have partition, ordering, TTL, compression and other parameters.
- ClickHouse supports replication and sharding.

BuzzHouse issues

- Most of the queries (>95%) still fail, mostly due to typing issues.
- Tables are not large enough to benchmark performance.
- Some of the oracles may give false positives, then disabled by default.
- Probabilities are static.
- No `SELECT 1 FROM idontexist;` queries.
- No code coverage, although this makes query generation slower.

More design questions

- What about running clients in parallel?
- Fuzzing the server? Distributed setting?
- Size of the queries generated?
- Quality of the error messages thrown?
- For how long a table should stay in the catalog?
- Detect slow queries in a legitimate way?

The verdict

- There won’t be a fuzzer that finds all the issues.
- Fuzzers follow the law of diminishing return.
- I can add more features to BuzzHouse but:
  - The codebase becomes more complex, then more subjective to bugs.
  - Finding issues in fuzzers is not trivial, because it’s not generated output.
  - More combinations, means less probability to find corner cases, then new bugs.

What's the solution?

- Use multiple fuzzers to test different properties of a database.
- Build invariants with oracles to find more issues.
- Reuse code between them if possible, eg. use BuzzHouse extensive CREATE TABLE options with AST fuzzer.
- Use a separate testing script for the server.

Use a combination of fuzzers

- AFL and libFuzzer for code coverage-guided fuzzing.
- SQLsmith (https://github.com/anse1/sqlsmith) for large query generation.
- SQLancer (https://github.com/sqlancer/sqlancer) for query correctness.
- Pstress (https://github.com/Percona-QA/pstress) for heavy load.
- Sysbench (https://github.com/akopytov/sysbench) for long workloads (>1h).
- AST fuzzer to mutate queries from tests, or inputs from other fuzzers.
- BuzzHouse for randomly generated catalog-backed queries.

Details at blog post

- More information at the engineering blog post: https://clickhouse.com/blog/buzzhouse-bridging-the-database-fuzzing-gap-for-testing-clickhouse

Blog / Engineering

BuzzHouse: Bridging the database fuzzing gap for testing ClickHouse

Pedro Ferreira
Jan 21, 2025 - 11 minutes read

Fuzzing has become a scorching research topic in the last few years to find issues in software, including crashes, bad output, and security vulnerabilities. Databases are no exception, and many research tools have been developed.

ClickHouse is also actively tested with fuzzers - over the years, several fuzzers, including SQLancer, SQLsmith, AST fuzzer, and, more recently, WINGFUZZ fuzzer, have been used to test ClickHouse.

Since I joined ClickHouse, I have been reviewing the existing testing infrastructure in ClickHouse and noticed a notorious gap in their fuzzers. None of them was capable of generating a wide complexity of queries while keeping query correctness in mind.

ClickHouse

Dinner & Drinks

FOSDEM Dinner with ClickHouse

Brussels Belgium, Saturday Feb 1st from 19:00-23:00pm

L'Ultime Atome
Rue Saint-Boniface 14, 1050 Ixelles, Belgium

ClickHouse

Thank you
Time for Q&A