DiffStream: Differential Output Testing for Stream Processing Programs

Presenting: Caleb Stanford
Coauthors: Konstantinos Kallas, Filip Niksic*, Rajeev Alur

*now at Google
Stream Processing

Streaming Data → Real-Time Decisions → Applications

Companies
- databricks
- Materialize
- aws KINESIS

Systems (DSPS)
- Spark
- Apache Flink
- APACHE STORM™
Evolution of Stream Processing Systems

Batch Processing
- Distributed, fault-tolerant
- High throughput

⇒

Stream Processing (DSPS)
- Distributed, fault-tolerant
- High throughput
- Low latency
- Stateful, event-by-event processing

MapReduce

→

Spark (Spark Streaming)

Apache Flink
Programming Support for DSPS

Achieving correct parallelization is difficult and prone to errors 🐜

Verification?

Testing?

Not accessible to end users

Not explored in the streaming setting -- state of the art is unit and integration testing

[Vianna, Ferreira, Gama 2019]
DiffStream: Differential Testing for Flink

1. Differential testing framework

Input
 random or manual

P1
 e.g. sequential

P2
 e.g. parallel

Matching algorithm

Specification
(ordering requirements)

Verdict

2. Core technical challenge

3. Implementation:  https://github.com/fniksic/diffstream
Programming Errors: Example

Topic Counting

Input:

... Startup ... Playoffs Apache Senator ...

User Queries

Pre-trained model

Output:

“Most frequent topic per region since the last query”
Programming Errors: Example

Not straightforward to parallelize!
- Due to

Not straightforward to test (e.g. unit testing)
- Correct program is a combination of ordered and out-of-order data

“Most frequent topic per region since the last query”
Programming Errors: Example

Parallelization attempts:

(Sequential solution):

Attempt 1 (automatic parallel):

Attempt 2:

Attempt 3:

Q: How to check correctness?
Q: How to specify correct ordering behavior?

Out of order data may or may not indicate a bug.
Our Solution: Differential Testing

Requirements:
- Accessible to end-users (no complex specifications)
- Combination of ordered and out-of-order data

Domain-specific insight: sequential program can act as spec!

Lightweight: only need to specify allowed ordering behavior

Black-box: avoids false positives due to underspecification
Lightweight Specification Language

Typical scenario: ordered by key, unordered across keys

Specification of allowed orderings:

$\text{spec: Event} \times \text{Event} \rightarrow \text{Bool}$

Typical examples:

- $(ev1, ev2) \rightarrow ev1.key == ev2.key$
- $(ev1, ev2) \rightarrow \text{False}$
- $(ev1, ev2) \rightarrow \text{True}$
- $|ev1.loc - ev2.loc| < 1$

Differential Testing Problem: is $S1 \equiv S2$ up to reorderings allowed in spec?
Technical Challenge: Checking Equivalence

Is $S_1 \equiv S_2$ equivalent up to the reorderings allowed in the specification?

Streams could be very large (real data or system under load)
Technical Challenge: Checking Equivalence

Is $S_1 \equiv S_2$ equivalent up to the reorderings allowed in the specification?

- **Naive solution**: store entire streams
- When can the algorithm drop events?

Streams could be very large (real data or system under load)
Checking Equivalence: Theorems

*Online* algorithm:
- Should reach verdict as soon as possible (correctness)
- Should use minimal space (optimality)

**Optimality Theorem:** Any other correct algorithm uses at least as much space.

**Discussion in the paper:** Space bounds in practice
Experiments: Overview

4 case studies

● Finding bugs in Flink applications
  ○ Aggregation over taxi GPS data
  ○ Parallel topic classification

● Existing programs: MapReduce
  ○ Differential testing avoids false positives!

● Testing overhead: What is the cost of online testing?
  ○ 5% reduction in maximum throughput
Case Study: MapReduce programs

ICSE 2014: 507 production jobs

- Existing work: false positives on non-commutative reducers

Diagnosis of false positives: Underspecification of input behavior and (in streaming case) possible orderings

Advantage of black box approach (differential testing):
- Detect 5/5 true positives
- Avoid 4/4 false positives
Future Opportunities

**Broader Goal:** make distributed programming over data streams more reliable by eliminating bugs due to unexpected behavior in the presence of parallelism.

Not a solved problem:
- **MapReduce:** commutative/associative aggregators
- **Static Verification:** not directly accessible to end users
- **This work:** bugs due to out-of-order output
  - Faults? Approximate computations?

**Rich opportunity for research!**
Summary

**DiffStream**: differential testing of streams in DSPS up to lightweight reordering specifications

- Online
- Optimal
- Effective + low overhead in practice

[https://github.com/fniksic/diffstream](https://github.com/fniksic/diffstream)
Supplementary Slides
Online Testing Overhead

5% overhead in maximum throughput

2 μs average latency

Yahoo Streaming Benchmark (2016)