Interactive and Automated Debugging for Big Data Analytics

Professor Miryung Kim
University of California, Los Angeles
Software Engineering and Analysis Lab at UCLA

Interactive Code Review
Refactoring and Transformation

Program Comprehension
Debugging

Test Debugging View
Atomic Change View
Data Science *elevating* Software Engineering

**Software Refactoring**
- Refactoring Field Study
- Quantifying Refactoring Cost and Benefits
- Impact on Regression Testing
- Role of API Refactoring

**API Evolution**
- Role of API Refactoring
- API Stability

**Empirical Studies of Software Changes**

**Code Redundancy**
- Clone genealogy
- Copy and paste practices
- Long lived clones
- Software forking and code porting

**Software Patches**
- Supplementary patches
- Omission errors
Data Science *elevating* Software Engineering

**Empirical Studies of Software Changes**
- Code Redundancy
  - Clone genealogy
  - Copy and paste practices
  - Long lived clones
  - Software forking and code porting
- Software Patches
  - Supplementary patches
  - Omission errors

**Automated and Interactive Software Dev Tools**
- Logical Program Differencing
  - LSdiff
  - Vdiff for VHDL
- Refactoring Reconstruction
  - Reffinder
- API Usage Adaptation
  - LibSync6
  - AURA
  - API Matching
- Interactive Code Review
  - Critics
- Transplantation and Test Reuse
  - Grafter
- Clone Removal Refactoring
  - RASE

**Software Refactoring**
- Refactoring Field Study
- Quantifying Refactoring Cost and Benefits
- Impact on Regression Testing
- Role of API Refactoring

**API Evolution**
- Role of API Refactoring
- API Stability
Data Science *elevating* Software Engineering

**Empirical Studies of Software Changes**

- **Software Refactoring**
  - Refactoring Field Study
  - Quantifying Refactoring Cost and Benefits
  - Impact on Regression Testing
  - Role of API Refactoring

- **API Evolution**
  - Role of API Refactoring
  - API Stability

**Program Transformation from Examples**

- Sydit
- LASE
- Cookbook

**Bug Finding**

- Refactoring Bugs
- Cloning Inconsistencies
- Fault Tracer
- Modularity Violations
- Prioritizing Tests for Refactoring

**Code Redundancy**

- Clone genealogy
- Copy and paste practices
- Long lived clones
- Software forking and code porting

**Software Patches**

- Supplementary patches
- Omission errors

**Logical Program Differencing**

- Lsdiff
- Vdiff for VHDL

**Refactoring Reconstruction**

- Reffinder

**API Usage Adaptation**

- LibSync6
- AURA
- API Matching

**Interactive Code Review**

- Critics

**Transplantation and Test Reuse**

- Grafter

**Clone Removal Refactoring**

- RASE

**Recommendation Systems**

**Automated and Interactive Software Dev Tools**
Current Research Focus: Software Engineering *elevating* Data Science

**Data Scientists in Software Teams**
- Background
- Work Activities
- Challenges
- Best Practices
- Quality Assurance

**SE Tools for Big Data Analytics**
- Interactive Debugger
- Data Provenance
- Automated Debugging
The Emerging Roles of Data Scientists on Software Teams

We are at a tipping point where there are large scale telemetry, machine, process and quality data.

Data scientists are emerging roles in SW teams due to an increasing demand for experimenting with real users and reporting results with statistical rigor.

We have conducted the first in-depth interview study and the largest scale survey of professional data scientists to characterize working styles.
Methodology for Studying “Data Scientists”

In-Depth Interviews [ICSE 2016]

16 data scientists
- 5 women and 11 men from eight different Microsoft organizations

Snowball sampling
- data-driven engineering meet-ups and technical community meetings
- word of mouth

Coding with Atlas.Ti
Clustering of participants

Survey [TSE 2018]

793 responses
- full-time data scientists
- employees with interest in data science

Questions about
- demographics
- skills
- self-perception
- working styles
- time spent
- challenges and best practices
Background of Data Scientists

Most CS, many **interdisciplinary** backgrounds

Many have **higher education** degrees

Survey: 41% have master’s degrees, and 22% have PhDs

Strong passion for data

“’I’ve always been a data kind of guy. I love playing with data. I’m very focused on how you can organize and make sense of data and being able to find patterns. I love patterns.”

Machine learning hackers. Need to know stats

“’My people have to know statistics. They need to be able to answer sample size questions, design experiment questions, know standard deviations, p-value, confidence intervals, etc.”
**Background of Data Scientists**

**PhD training** contributes to working style

“*It has never been, in my four years, that somebody came and said, “Can you answer this question?” I mostly sit around thinking, “How can I be helpful?” Probably that part of your PhD is you are figuring out what is the most important questions.”* [P13]

“I have a PhD in experimental physics, so pretty much, I am used to designing experiments.” [P6]

“*Doing data science is kind of like doing research. It looks like a good problem and looks like a good idea. You think you may have an approach, but then maybe you end up with a dead end.”* [P5]
Time Spent on Activities

Hours spent on certain activities (self reported, survey, N=532)
Time Spent on Activities

Cluster analysis on relative time spent (k-means)

532 data scientists at Microsoft

Clustering based on relative time spent in activities
9 Distinct Categories of Data Scientists based on Work Activities

Data Scientists in Software Teams: State of the Art and Challenges, Kim et al. IEEE Transactions on Software Engineering
## Category: Data Shaper

<table>
<thead>
<tr>
<th>Category</th>
<th>Entire Population 532 People</th>
<th>Cluster 1 Polymath 156 People</th>
<th>Cluster 2 Data Evangelist 71 People</th>
<th>Cluster 3 Data Preparer 122 People</th>
<th>Cluster 4 Data Shaper 33 People</th>
<th>Cluster 5 Data Analyzer 24 People</th>
<th>Cluster 6 Platform Builder 27 People</th>
<th>Cluster 7 Moonlighter 50% 63 People</th>
<th>Cluster 8 Moonlighter 10% 32 People</th>
<th>Cluster 9 Act on Insight 4 People</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12.0% 4.7h</td>
<td>10.4% 4.4h</td>
<td>6.8% 2.2h</td>
<td>24.5% 9.4h</td>
<td>5.6% 2.5h</td>
<td>9.9% 3.7h</td>
<td>12.5% 4.4h</td>
<td>7.3% 3.1h</td>
<td>2.9% 1.2h</td>
<td>0.9% 0.1h</td>
</tr>
<tr>
<td></td>
<td>7.2% 2.9h</td>
<td>8.5% 4.0h</td>
<td>2.1% 0.7h</td>
<td>4.9% 1.9h</td>
<td>1.8% 0.7h</td>
<td>0.9% 0.3h</td>
<td>48.5% 18.4h</td>
<td>5.0% 2.2h</td>
<td>1.4% 0.6h</td>
<td>2.1% 0.2h</td>
</tr>
<tr>
<td></td>
<td>11.7% 4.9h</td>
<td>11.5% 5.1h</td>
<td>7.7% 2.9h</td>
<td>19.6% 7.8h</td>
<td>27.0% 11.5h</td>
<td>5.8% 2.4h</td>
<td>6.1% 2.6h</td>
<td>5.0% 2.1h</td>
<td>1.9% 0.8h</td>
<td>0.9% 0.1h</td>
</tr>
<tr>
<td></td>
<td>12.6% 5.2h</td>
<td>15.1% 6.7h</td>
<td>7.7% 2.9h</td>
<td>10.0% 4.0h</td>
<td>25.7% 10.9h</td>
<td>2.4% 0.9h</td>
<td>4.3% 1.9h</td>
<td>2.8% 1.2h</td>
<td>1.6% 0.7h</td>
<td>1.8% 0.2h</td>
</tr>
<tr>
<td></td>
<td>4.8% 2.1h</td>
<td>9.1% 4.0h</td>
<td>7.7% 2.9h</td>
<td>3.0% 1.3h</td>
<td>6.0% 2.6h</td>
<td>4.6% 1.8h</td>
<td>3.8% 1.1h</td>
<td>3.8% 1.2h</td>
<td>0.4% 0.2h</td>
<td>0.9% 0.1h</td>
</tr>
<tr>
<td></td>
<td>6.0% 3.0h</td>
<td>7.7% 3.6h</td>
<td>7.0% 2.6h</td>
<td>10.0% 4.1h</td>
<td>8.9% 3.8h</td>
<td>6.6% 2.7h</td>
<td>2.7% 1.2h</td>
<td>2.7% 1.2h</td>
<td>1.5% 0.6h</td>
<td>1.5% 0.2h</td>
</tr>
<tr>
<td></td>
<td>8.5% 3.5h</td>
<td>7.4% 3.5h</td>
<td>12.0% 4.5h</td>
<td>11.6% 4.1h</td>
<td>7.6% 3.3h</td>
<td>5.2% 2.0h</td>
<td>4.4% 1.9h</td>
<td>4.2% 1.8h</td>
<td>1.7% 0.8h</td>
<td>1.7% 0.3h</td>
</tr>
<tr>
<td></td>
<td>0.2% 0.1h</td>
<td>7.9% 3.6h</td>
<td>23.0% 8.6h</td>
<td>8.8% 3.5h</td>
<td>7.5% 3.2h</td>
<td>5.8% 2.4h</td>
<td>4.1% 1.9h</td>
<td>7.8% 3.3h</td>
<td>2.3% 0.8h</td>
<td>2.1% 0.3h</td>
</tr>
<tr>
<td></td>
<td>2.4% 1.1h</td>
<td>3.2% 1.5h</td>
<td>3.7% 1.3h</td>
<td>1.5% 0.7h</td>
<td>2.1% 1.0h</td>
<td>1.8% 0.9h</td>
<td>1.0h 0.4h</td>
<td>5.9% 1.8h</td>
<td>0.6% 0.3h</td>
<td>2.1% 0.2h</td>
</tr>
<tr>
<td></td>
<td>2.1% 0.9h</td>
<td>5.2% 2.3h</td>
<td>13.4% 6.0h</td>
<td>9.5% 3.9h</td>
<td>3.3% 1.3h</td>
<td>4.2% 1.6h</td>
<td>1.4h 0.6h</td>
<td>2.5% 1.0h</td>
<td>1.4% 0.6h</td>
<td>2.9% 0.3h</td>
</tr>
<tr>
<td></td>
<td>5.5% 2.1h</td>
<td>4.0% 2.0h</td>
<td>6.0% 2.6h</td>
<td>1.5% 0.7h</td>
<td>3.3% 1.3h</td>
<td>2.8% 1.3h</td>
<td>1.1h 0.4h</td>
<td>1.9% 0.7h</td>
<td>1.4% 0.6h</td>
<td>36.1% 11.8h</td>
</tr>
<tr>
<td></td>
<td>4.1% 1.9h</td>
<td>10.1% 4.0h</td>
<td>5.7% 2.3h</td>
<td>9.6% 3.9h</td>
<td>5.7% 2.3h</td>
<td>3.2% 1.2h</td>
<td>3.4% 1.1h</td>
<td>1.8% 0.7h</td>
<td>80.9% 30.8h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>15.1% 6.7h</td>
<td>9.5% 3.9h</td>
<td>5.7% 2.3h</td>
<td>13.4% 6.0h</td>
<td>6.0% 2.6h</td>
<td>4.2% 1.6h</td>
<td>1.1h 0.4h</td>
<td>1.9% 0.7h</td>
<td>14.1% 4.8h</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Category: Platform Builder

<table>
<thead>
<tr>
<th>Cluster 6 Platform Builder</th>
<th>12.5%</th>
<th>48.5%</th>
<th>6.1%</th>
<th>4.3%</th>
<th>3.8%</th>
<th>2.7%</th>
<th>4.4%</th>
<th>4.1%</th>
<th>2.1%</th>
<th>3.0%</th>
<th>1.4%</th>
<th>6.9%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster 7 Moonlyghter 50%-63 people</td>
<td>7.3%</td>
<td>5.0%</td>
<td>5.0%</td>
<td>5.5%</td>
<td>2.8%</td>
<td>4.2%</td>
<td>7.8%</td>
<td>5.9%</td>
<td>1.8%</td>
<td>5.7%</td>
<td>2.5%</td>
<td>40.5%</td>
</tr>
<tr>
<td>Cluster 8 Moonlyghter 10%-32 people</td>
<td>2.9%</td>
<td>1.4%</td>
<td>1.9%</td>
<td>1.6%</td>
<td>0.4%</td>
<td>1.5%</td>
<td>1.7%</td>
<td>2.3%</td>
<td>0.6%</td>
<td>2.1%</td>
<td>2.9%</td>
<td>80.9%</td>
</tr>
<tr>
<td>Cluster 9 Act on Insight-4 people</td>
<td>0.9%</td>
<td>2.1%</td>
<td>1.6%</td>
<td>0.9%</td>
<td>1.5%</td>
<td>3.8%</td>
<td>10.1%</td>
<td>3.0%</td>
<td>57.1%</td>
<td>11.8%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Platform Builder**

- **Back End Programming:** 70% vs. 36%
- **Big and Distributed Data:** 81% vs. 50%
- **Classic Statistics:** 30% vs. 50%
- **Front End Programming:** 63% vs. 31%
- **SQL:** 89% vs. 68%
- **C/C++/C#:** 70% vs. 45%
Challenges that Data Scientists Face

**Poor data quality**

“Poor data quality. This combines with the expectation that as an analyst, this is your job to fix (or even your fault if it exists), not that you are the main consumer of this poor quality data.” [P754]

**Batch jobs**

“Because of the huge data size, batch processing jobs like Hadoop make iterative work expensive and quick visualization of large data painful.” [P651]
Challenges in Ensuring “Correctness”

**Validation** is a major challenge.

“Honestly, we don’t have a good method for this.” [P457]

“Just because the math is right, doesn’t mean that the answer is right.” [P307]

“When it comes to data, trust nothing.” [P59]

**Explainability** is important. Participants warned about overreliance on aggregate metrics—“to gain insights, you must go one level deeper.”

“Interpreting [data] without knowing why it looks like it does will most likely lead you into a wrong direction.” [P577]
Big Data Debugging in the Dark

Develop locally → Hope it works → Run in cloud → Bug!

Guesswork

Tools:
- Google Map Reduce
- Hadoop
- Spark
- Hive
Software Engineering for Data Science

Data Scientists in Software Teams
- Background
- Work Activities
- Challenges
- Best Practices
- Quality Assurance

SE Tools for Big Data Analytics
- Interactive Debugger
- Data Provenance
- Automated Debugging
BigDebug: Debugging Primitives for Interactive Big Data Processing in Spark

Muhammad Ali Gulzar, Matteo Interlandi, Seunghyun Yoo, Sai Deep Tetali, Tyson Condie, Todd Millstein, Miryung Kim

[ICSE 2016, FSE Tool Demo 2016, SIGMOD Tool Demo 2017]
Running a Map Reduce Job on Cluster

A user submits a job

A job is distributed to workers in cluster

Each worker performs pipelined transformations on a partition with millions of records
Motivating Scenario: Election Record Analysis

- Alice writes a Spark program that runs correctly on local machine (100MB data) but crashes on cluster (1TB)
- Alice cannot see the crash-inducing intermediate result.
- Alice cannot identify which input from 1TB causing crash
- When crash occurs, all intermediate results are thrown away.

```
val log = "s3n://poll.log"
val text_file = spark.textFile(log)
val count = text_file
  .filter(line => line.split()[3].toInt > 1440012701)
  .map(line => (line.split()[1], 1))
  .reduceByKey(_ + _).collect()
```

Task 31 failed 3 times; aborting job
ERROR Executor: Exception in task 31 in stage 0 (TID 31)
java.lang.NumberFormatException
Why Traditional Debug Primitives Do Not Work for Apache Spark?

Enabling interactive debugging requires us to **re-think the features of traditional debugger** such as GDB

- Pausing the entire computation on the cloud could reduce throughput
- It is clearly infeasible for a user to inspect billion of records through a regular watchpoint
- Even launching remote JVM debuggers to individual worker nodes cannot scale for big data computing
1. Simulated Breakpoint

Simulated breakpoint replays computation from the latest materialization point where data is stored in memory
1. Simulated Breakpoint – Realtime Code Fix

Allow a user to fix code after the breakpoint
2. On-Demand Guarded Watchpoint

Watchpoint captures individual data records matching a user-provided guard

```java
state.equals("TX") || state.equals("CA")
```
A user can either correct the crashed record, skip the crash culprit, or supply a code fix to repair the crash culprit.
4. Backward and Forward Tracing

A user can also issue tracing queries on intermediate records at realtime
Demo: BigDebug Interactive Debugger
[FSE 2016 Demo, SIGMOD 2017 Demo]
Q1 : How does BigDebug scale to massive data?

BigDebug retains scale up property of Spark. This property is critical for Big Data processing frameworks.
Q2 : What is the performance overhead of debugging primitives?

<table>
<thead>
<tr>
<th>Program</th>
<th>Dataset size (GB)</th>
<th>Max</th>
<th>Max w/o Latency Alert</th>
<th>Watchpoint</th>
<th>Crash Culprit</th>
<th>Tracing</th>
</tr>
</thead>
<tbody>
<tr>
<td>WordCount</td>
<td>0.5 - 1000</td>
<td>2.5X</td>
<td>1.34X</td>
<td>1.09X</td>
<td>1.18X</td>
<td>1.22X</td>
</tr>
<tr>
<td>Grep</td>
<td>20 - 90</td>
<td>1.76X</td>
<td>1.07X</td>
<td>1.05X</td>
<td>1.04X</td>
<td>1.05X</td>
</tr>
<tr>
<td>PigMix-L1</td>
<td>1 - 200</td>
<td>1.38X</td>
<td>1.29X</td>
<td>1.03X</td>
<td>1.19X</td>
<td>1.24X</td>
</tr>
</tbody>
</table>

Max : All the features of BigDebug are enabled

BigDebug poses at most 2.5X overhead with the maximum instrumentation setting.
Titian: Data Provenance Support in Spark

Matteo Interlandi, Kshitij Shah, Sai Deep Tetali, Muhammad Ali Gulzar, Seunghyun Yoo, Miryung Kim, Todd Millstein, Tyson Condie
[VLDB 2016]
Data Provenance – Example in SQL

SELECT time, AVG(temp)
FROM sensors
GROUP BY time

Outlier

Why ID-2 and ID-3 have those high values?
Step 1: Instrumented Workflow in Spark

**Stage 1**
- Hadoop LineageRDD
  - lines
  - errors
  - codes
  - pairs

**Input ID** | **Output ID**
---|---
offset1 | id1
offset2 | id2
offset3 | id3

**Stage 2**
- Reducer LineageRDD
  - counts
  - reports

**Input ID** | **Output ID**
---|---
[p1, p2] | 400
[p1] | 4

**LineageRDD**
- Input ID | Output ID
---|---
{id1, id3} | 400
{id2} | 4
400 | id1
4 | id2

**Reports**
- Stage LineageRDD
Step 2: Example Backward Tracing

**Hadoop**

<table>
<thead>
<tr>
<th>Input ID</th>
<th>Output ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset1</td>
<td>id1</td>
</tr>
<tr>
<td>offset2</td>
<td>id2</td>
</tr>
<tr>
<td>offset3</td>
<td>id3</td>
</tr>
</tbody>
</table>

**Combiner**

<table>
<thead>
<tr>
<th>Input ID</th>
<th>Output ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ id1, id3}</td>
<td>400</td>
</tr>
<tr>
<td>{ id2}</td>
<td>4</td>
</tr>
</tbody>
</table>

**Reducer**

<table>
<thead>
<tr>
<th>Input ID</th>
<th>Output ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>[p1, p2]</td>
<td>400</td>
</tr>
<tr>
<td>[ p1 ]</td>
<td>4</td>
</tr>
</tbody>
</table>

**Stage**

<table>
<thead>
<tr>
<th>Input ID</th>
<th>Output ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>id1</td>
</tr>
<tr>
<td>4</td>
<td>id2</td>
</tr>
</tbody>
</table>
Step 2: Example Backward Tracing

<table>
<thead>
<tr>
<th>Hadoop</th>
<th>Combiner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input ID</td>
<td>Output ID</td>
</tr>
<tr>
<td>offset1</td>
<td>id1</td>
</tr>
<tr>
<td>offset2</td>
<td>id2</td>
</tr>
<tr>
<td>offset3</td>
<td>id3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input ID</th>
<th>Output ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ id1, id 3}</td>
<td>400</td>
</tr>
<tr>
<td>{ id2 }</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input ID</th>
<th>Output ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>400</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hadoop</th>
<th>Combiner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input ID</td>
<td>Output ID</td>
</tr>
<tr>
<td>offset1</td>
<td>id1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input ID</th>
<th>Output ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ id1, ...}</td>
<td>400</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input ID</th>
<th>Output ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>400</td>
</tr>
</tbody>
</table>
Step 2: Example Backward Tracing

**Hadoop**

<table>
<thead>
<tr>
<th>Input ID</th>
<th>Output ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset1</td>
<td>id1</td>
</tr>
<tr>
<td>offset2</td>
<td>id2</td>
</tr>
<tr>
<td>offset3</td>
<td>id3</td>
</tr>
</tbody>
</table>

**Combiner**

<table>
<thead>
<tr>
<th>Input ID</th>
<th>Output ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ id1, id 3}</td>
<td>400</td>
</tr>
<tr>
<td>{ id2 }</td>
<td>4</td>
</tr>
</tbody>
</table>

Hadoop.Output ID ⊗ Combiner.Input ID

Hadoop and Combiner are connected by a backward tracing mechanism.
Automated Debugging in Data Intensive Scalable Computing

Muhammad Ali Gulzar, Matteo Interlandi, Xueyuan Han, Mingda Li
Tyson Condie, Miryung Kim
[SOCC 2017]
Motivating Example

• Alice writes a Spark program that identifies, for each state in the US, the delta between the minimum and the maximum snowfall reading for each day of any year and for any particular year.

• An input data record that measures 1 foot of snowfall on January 1st of Year 1992, in the 99504 zip code (Anchorage, AK) area, appears as

99504, 01/01/1992, 1ft
Problem Definition

- Using a test function, a user can specify incorrect results

```python
def test(key: String, delta: Float) : Boolean = {
delta < 6000
}
```

Given a test function, the goal is to identify a minimum subset of the input that is able to reproduce the same test failure.
Existing Approach 1: Data Provenance for Spark

TextFile → FlatMap → GroupByKey → Map → Output

99504, 01/01/1992, 1ft
99504, 03/01/1992, 0.1ft
99504, 01/01/1993, 70in
99504, 03/01/1993, 145mm
99504, 01/01/1994, 245mm
99504, 01/01/1993, 85mm
90031, 02/01/1991, 0mm

AK, 01/01, 304.8
AK, 1992, 304.8
AK, 03/01, 30.5
AK, 1992, 30.5
AK, 01/01, 21336
AK, 1993, 21336
AK, 03/01, 145
AK, 1993, 145
AK, 01/01, 245
AK, 1994, 245

AK, 01/01, [304.8, 21336, 245, 85]
AK, 03/01, [30.5, 145]
AK, 1992, [304.8, 30.5]
AK, 1993, [21336, 145, 85]
AK, 1994, [245]
CA, 02/01, [0]
CA, 1991, [0]

AK, 01/01, 21251
AK, 03/01, 114.5
AK, 1992, 274.3
AK, 1993, 21251
AK, 1994, 0
CA, 02/01, 0
CA, 1991, 0

It over-approximates the scope of failure-inducing inputs i.e. records in the faulty key-group are all marked as faulty
Existing Approach 2: Delta Debugging

- Delta Debugging performs a systematic binary search-like procedure on the input dataset using a test oracle function.

It does not prune input records known to be irrelevant because of the lack of semantic understanding of data-flow operators.
Existing Approach 2: Delta Debugging

- Delta Debugging performs a systematic binary search-like procedure on the input dataset using a test oracle function.

**Run 2**

It does not prune input records known to be irrelevant because of the lack of semantic understanding of data-flow operators.
Existing Approach 2: Delta Debugging

- Delta Debugging performs a systematic binary search-like procedure on the input dataset using a test oracle function.

Run 3

It does not prune input records known to be irrelevant because of the lack of semantic understanding of data-flow operators.
Existing Approach 2: Delta Debugging

- Delta Debugging performs a systematic binary search-like procedure on the input dataset using a test oracle function.

It does not prune input records known to be irrelevant because of the lack of semantic understanding of data-flow operators.
Existing Approach 2: Delta Debugging

- Delta Debugging performs a systematic binary search-like procedure on the input dataset using a test oracle function.

TextFile → FlatMap → GroupByKey → Map → Output

99504, 01/01/1992, 1ft
99504, 03/01/1992, 0.1ft
99504, 01/01/1993, 70in
AK, 01/01, 304.8
AK, 1992, 304.8
AK, 1992, [304.8]

Run 5

It does not prune input records known to be irrelevant because of the lack of semantic understanding of data-flow operators.
Existing Approach 2: Delta Debugging

- Delta Debugging performs a systematic binary search-like procedure on the input dataset using a test oracle function

It does not prune input records known to be irrelevant because of the lack of semantic understanding of data-flow operators
**Existing Approach 2: Delta Debugging**

- Delta Debugging performs a systematic binary search-like procedure on the input dataset using a test oracle function

Run 7

- It does not prune input records known to be irrelevant because of the lack of semantic understanding of data-flow operators
Existing Approach 2: Delta Debugging

- Delta Debugging performs a systematic binary search-like procedure on the input dataset using a test oracle function.

Run 8

It does not prune input records known to be irrelevant because of the lack of semantic understanding of data-flow operators.
Existing Approach 2: Delta Debugging

- Delta Debugging performs a systematic binary search-like procedure on the input dataset using a test oracle function.

It does not prune input records known to be irrelevant because of the lack of semantic understanding of data-flow operators.

Run 9
Automated Debugging in DISC with BigSift

Input: A Spark Program, A Test Function

Output: Minimum Fault-Inducing Input Records

Data Provenance + Delta Debugging

Test Predicate Pushdown

Prioritizing Backward Traces

Bitmap based Test Memoization
Optimization 1: Test Predicate Pushdown

- **Observation:** During backward tracing, data provenance traces through all the partitions even though only a few partitions are faulty

If applicable, BigSift pushes down the test function to test the output of combiners in order to isolate the faulty partitions.
Optimization 2: Prioritizing Backward Traces

- **Observation**: The same faulty input record may contribute to multiple output records failing the test.

In case of multiple faulty outputs, BigSift overlaps two backward traces to minimize the scope of fault-inducing input records.
Optimization 3: Bitmap Based Test Memoization

- **Observation:** Delta debugging may try running a program on the same subset of input redundantly.

- **BigSift leverages bitmap to compactly encode the offsets of original input to refer to an input subset**

We use a bitmap based test memoization technique to avoid redundant testing of the same input dataset.
## RQ1: Performance Improvement over Delta Debugging

<table>
<thead>
<tr>
<th>Subject Program</th>
<th>Fault</th>
<th>Running Time (sec)</th>
<th>Debugging Time (sec)</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Original Job</td>
<td>DD</td>
<td>BigSift</td>
</tr>
<tr>
<td>Movie Histogram</td>
<td>Code</td>
<td>56.2</td>
<td>232.8</td>
<td>17.3</td>
</tr>
<tr>
<td>Inverted Index</td>
<td>Code</td>
<td>107.7</td>
<td>584.2</td>
<td>13.4</td>
</tr>
<tr>
<td>Rating Histogram</td>
<td>Code</td>
<td>40.3</td>
<td>263.4</td>
<td>16.6</td>
</tr>
<tr>
<td>Sequence Count</td>
<td>Code</td>
<td>356.0</td>
<td>13772.1</td>
<td>208.8</td>
</tr>
<tr>
<td>Rating Frequency</td>
<td>Code</td>
<td>77.5</td>
<td>437.9</td>
<td>14.9</td>
</tr>
<tr>
<td>College Student</td>
<td>Data</td>
<td>53.1</td>
<td>235.3</td>
<td>31.8</td>
</tr>
<tr>
<td>Weather Analysis</td>
<td>Data</td>
<td>238.5</td>
<td>999.1</td>
<td>89.9</td>
</tr>
<tr>
<td>Transit Analysis</td>
<td>Code</td>
<td>45.5</td>
<td>375.8</td>
<td>20.2</td>
</tr>
</tbody>
</table>

BigSift provides up to a 66X speed up in isolating the precise fault-inducing input records, in comparison to the baseline DD.
RQ2: Debugging Time vs. Original job time

<table>
<thead>
<tr>
<th>Subject Program</th>
<th>Fault</th>
<th>Running Time (sec) Original Job</th>
<th>Running Time (sec) DD</th>
<th>Running Time (sec) BigSift</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Movie Histogram</td>
<td>Code</td>
<td>56.2</td>
<td>232.8</td>
<td>17.3</td>
<td>13.5X</td>
</tr>
<tr>
<td>Inverted Index</td>
<td>Code</td>
<td>107.7</td>
<td>584.2</td>
<td>13.4</td>
<td>43.6X</td>
</tr>
<tr>
<td>Rating Histogram</td>
<td>Code</td>
<td>40.3</td>
<td>263.4</td>
<td>16.6</td>
<td>15.9X</td>
</tr>
<tr>
<td>Sequence Count</td>
<td>Code</td>
<td>356.0</td>
<td>13772.1</td>
<td>208.8</td>
<td>66.0X</td>
</tr>
<tr>
<td>Rating Frequency</td>
<td>Code</td>
<td>77.5</td>
<td>437.9</td>
<td>14.9</td>
<td>29.5X</td>
</tr>
<tr>
<td>College Student</td>
<td>Data</td>
<td>53.1</td>
<td>235.3</td>
<td>31.8</td>
<td>7.4X</td>
</tr>
<tr>
<td>Weather Analysis</td>
<td>Data</td>
<td>238.5</td>
<td>999.1</td>
<td>89.9</td>
<td>11.1X</td>
</tr>
<tr>
<td>Transit Analysis</td>
<td>Code</td>
<td>45.5</td>
<td>375.8</td>
<td>20.2</td>
<td>18.6X</td>
</tr>
</tbody>
</table>

On average, BigSift takes 62% less time to debug a single faulty output than the time taken for a single run on the entire data.
RQ2: Debugging Time

On average, BigSift takes 62% less time to debug a single faulty output than the time taken for a single run on the entire data.
RQ3: Fault Localizability over Data Provenance

BigSift leverages DD after DP to continue fault isolation, achieving several orders of magnitude $10^3$ to $10^7$ better precision.
Summary: Debugging Big Data Analytics

- **Easy to use** interactive debugger by re-defining debug primitives for big data cloud computing

- **Visibility of data** into running workflow **by tracking data provenance**

- **Automated fault localization for big data cloud computing** that provides $10^3 \times - 10^7 \times$ more precision than data provenance in terms of fault localizability and up to 66X speed up in debugging time over baseline Delta Debugging.
Software Engineering *elevating* Data Science

**Data Scientists in Software Teams**
- [ICSE ‘16, TSE ‘18]
  - Background
  - Work Activities
  - Challenges
  - Best Practices
  - Quality Assurance

**Debugging for Big Data Analytics**
- Interactive Debugger [ICSE ‘16]
- Data Provenance [VLDB ‘16]
- Automated Debugging [SoCC ‘17]

**Data Summary and Explanation**
- “How do we characterize data by inferring the underlying type and format?”

**Automated Testing for Big Data Analytics**
- “How do we help select (sample) data for local testing?”
- “How do we generate test data to achieve high code coverage?”
- Combine symbolic execution and the semantics of data flow operators

**Optimization for Iterative Development**
- “How can we re-compute big data analytics in case of code changes?” [SoCC ‘16]

**Late Stage Customization of Big Data System Stack**
- “How do we customize Big Data runtime for the actual use of big data analytics?”
Thanks to my collaborators

**UCLA on Big Data Debugging:** Muhammad Ali Gulzar, Tyson Condie, Matteo Interlandi, Mingda Li, Michael Han, Sai Deep Tetali, Todd Millstein

**Microsoft Research on Data Scientist Studies:** Tom Zimmermann, Andrew Begel, and Rob DeLine
Big Data needs awesome software engineering tools

Diagnose
- Debugging
- Intelligent sampling and testing
- Root cause analysis

Fix
- Data cleaning

Optimize
- Performance analytics
- Code analytics