Automated Debugging In Data Intensive Scalable Computing Systems

Muhammad Ali Gulzar\textsuperscript{1}, Matteo Interlandi\textsuperscript{3}, Xueyuan Han\textsuperscript{2}, Mingda Li\textsuperscript{1}, Tyson Condie\textsuperscript{1}, and Miryung Kim\textsuperscript{1}

\textsuperscript{1}University of California, Los Angeles
\textsuperscript{2}Harvard University
\textsuperscript{3}Mircrosoft
Big Data Debugging in the Dark

1. Develop locally
2. Hope it works
3. Run in cloud
4. Bug!
5. Guesswork

Map Reduce

Google
Hadoop
Spark
Hive
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.

<table>
<thead>
<tr>
<th>Zip Code</th>
<th>Date</th>
<th>Snow Fall</th>
</tr>
</thead>
<tbody>
<tr>
<td>99504</td>
<td>01/01/1994</td>
<td>245mm</td>
</tr>
<tr>
<td>99504</td>
<td>01/01/1993</td>
<td>85mm</td>
</tr>
<tr>
<td>90031</td>
<td>02/01/1991</td>
<td>0mm</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Problem Definition

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

Given a test function, the goal is to identify a minimum subset of the input that is able to reproduce the same test failure.

```scala
def test(key: String, delta: Float): Boolean = {
  delta < 6000
}
```
Existing Approach 1: Data Provenance for Spark

Existing Approach 1: Data Provenance for Spark

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-like search 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-like search on the input dataset using a test oracle function

```
<table>
<thead>
<tr>
<th>TextFile</th>
<th>FlatMap</th>
<th>GroupByKey</th>
<th>Map</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>99504, 01/01/1992, 1ft</td>
<td>AK, 01/01, 304.8</td>
<td>AK, 01/01, [304.8]</td>
<td>AK, 01/01, 0</td>
<td></td>
</tr>
<tr>
<td>99504, 03/01/1992, 0.1ft</td>
<td>AK, 1992, 304.8</td>
<td>AK, 03/01, [30.5]</td>
<td>AK, 03/01, 0</td>
<td></td>
</tr>
<tr>
<td>99504, 01/01/1993, 70in</td>
<td>AK, 03/01, 30.5</td>
<td>AK, 1992, [304.8, 30.5]</td>
<td>AK, 1992, 274.3</td>
<td></td>
</tr>
</tbody>
</table>
```

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-like search 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-like search 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-like search 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-like search 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-like search 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-like search 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.
Automated Debugging in DISC with BigSift

Input: A Spark Program, A Test Function

Data Provenance + Delta Debugging

Test Predicate Pushdown

Prioritizing Backward Traces

Bitmap based Test Memoization

Output: Minimum Fault-Inducing Input Records
Optimization 1: Test Predicate Pushdown

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

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 faulty output due to operators such as Join or Flatmap.

In case of multiple faulty outputs, BigSift overlaps two backward traces to minimize the scope of fault-inducing input records.
We use a bitmap based test memoization technique to avoid redundant testing of the same input dataset.

**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.

<table>
<thead>
<tr>
<th>Input Data</th>
<th>Bitmap</th>
<th>Test Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 1 1 1</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>0 1 0 1 0 0</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Evaluation Questions

- **RQ1**: How much improvement in the debugging time does BigSift provide in comparison to delta debugging?

- **RQ2**: How long is the debugging time of BigSift in comparison to original running time of a job?

- **RQ3**: How much improvement in the precision of fault-inducing input records does BigSift provide in comparison to data provenance?
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>Debugging Time (sec) DD</th>
<th>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 vs. Original Job 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.
BigSift leverages DD after DP to continue fault isolation, achieving several orders of magnitude $10^3$ to $10^7$ better precision.
Conclusion

- BigSift is the first piece of work in automated debugging of big data analytics in DISC.

- BigSift provides $10^3 \times - 10^7 \times$ more precision than data provenance in terms of fault localizability.

- It provides up to 66X speed up in debugging time over baseline Delta Debugging.

- In our evaluation we have observed that, on average, BigSift finds the faulty input in 62% less than the original job execution time.
Questions?